#4 Cert 2nd 01

Attorney Docket No. 826.1725

JC971 U.S. PTO 09/835623 04/17/01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:	)	
	)	
Hidetoshi IWASHITA	)	
	)	Group Art Unit: To be Assigned
Application No.: To be Assigned	)	
	)	Examiner: To be Assigned
Filed: April 17, 2001	)	
	)	

For: COMPILER SYSTEM, COMPILING METHOD, AND STORAGE MEDIUM FOR STORING COMPILING PROGRAM

SUBMISSION OF CERTIFIED COPY OF PRIOR FOREIGN APPLICATION IN ACCORDANCE WITH THE REQUIREMENTS OF 37 C.F.R. §1.55

Assistant Commissioner for Patents
Washington, D.C. 2023l

Sir:

In accordance with the provisions of 37 C.F.R. §1.55, the applicant(s) submit(s) herewith a certified copy of the following foreign application:

Japanese Patent Application No. PCT/JP99/00381
Filed: January 29, 2001

It is respectfully requested that the applicant(s) be given the benefit of the foreign filing date as evidenced by the certified papers attached hereto, in accordance with the requirements of 35 U.S.C. §119.

Respectfully submitted,

STAAS & HALSEY LLP

By:_____

James D. Halsey, Jr.
Registration No. 22,729

700 11th Street, N.W., Ste. 500
Washington, D.C. 20001
(202) 434-1500
Date: April 17, 2001

PATENT COOPERATION TREATY

Issuer: Japanese Patent Office (Receiving Office)

Applicant Agent

Yoshiyuki Osuga

Address
3rd Fl., Nibancho Bldg.
8-20, Nibancho, Chiyoda-ku, Tokyo
102-0084

NOTICE OF MAILING-OUT THE REQUESTED DOCUMENT

(The rule of the Law, Sec.37 & Sec.37-2)
[PCT rule 20.9, 22.1 (d)]

	Date of mailing (day/month/year) 13. 03. 01
Applicant's or agent's file reference 9805482	**Enclosure** Official Copy of the International Application
International application No. PCT/JP99/00381	**International filing date** (day/month/year) 29. 01. 99
Applicant FUJITSU LIMITED	

We, the Receiving Office, hereby confirm that "a set of the official copies of the documents associated with the international application at the time of the application and with its procedural supplements or its procedural amendments" as requested by the applicant is issued.

Full address for the Receiving Office Japanese Patent Office (RO/JP) 4-3, Kasumigaseki 3-chome Chiyoda-ku, Tokyo 100-8915 Japan (phone No. 03-3592-1308)	**Authorized Officer** Commissioner, Patent Office

Form PCT/RO/122 (July 1992)

We hereby certify that the accompanying tamper sealed copy represents the corresponding original documents.

Date: 12 March 2001

Seal of: Mr. Takayuki Kiyodera, Trade and Industry Officer

特　許　協　力　条　約

発信人　日本国特許庁（受理官庁）

出願人代理人

大菅　義之　　殿

あて名

〒１０２－００８４

東京都千代田区二番町８番地２０

二番町ビル３Ｆ

請求があった書類の送付通知書

（法施行規則第３７条、同第３７条の２）

〔ＰＣＴ規則２０．９、２２．１(d)〕

	発送日（日．月．年） １３．０３．０１
出願人又は代理人の書類記号 ９８０５４８２	国際出願の謄本又は写しの認証を同封
国際出願番号 ＰＣＴ／ＪＰ９９／００３８１	国際出願日（日．月．年） ２９．０１．９９
出願人（氏名又は名称） 富士通株式会社	

受理官庁は、出願人から請求のあった「出願時の国際出願に係わる書類又は手続の補完若しくは手続の補正に係わる書類の謄本」~~又は「国際出願の写しの認証」~~を交付する。

受理官庁の名称及びあて名 日　本　国　特　許　庁　（ＲＯ／ＪＰ） 郵便番号１００－８９１５ TEL０３－３５９２－１３０８ 日本国東京都千代田区霞が関三丁目４番３号	権限のある職員 特　許　庁　長　官

様式ＰＣＴ／ＲＯ／１２２（１９９２年７月）

受理官庁用写し

特許協力条約に基づく国際出願

願　　書

出願人は、この国際出願が特許協力条約に従って処理されることを請求する。

PCT/JP99/00381

受理官庁記入欄

国際出願番号	PCT/JP99/00381
国際出願日	29.01.99
(受付印)	PCT International Application 日本国特許庁
出願人又は代理人の書類記号 (希望する場合、最大12字)	9805482

第I欄　発明の名称

コンパイラ装置、コンパイル方法、およびそのためのプログラムを格納した記憶媒体

第II欄　出願人

氏名（名称）及びあて名：(姓・名の順に記載；法人は公式の完全な名称を記載；あて名は郵便番号及び国名も記載)

富士通株式会社
FUJITSU LIMITED

〒211－8588 日本国神奈川県川崎市中原区上小田中4丁目1番1号
1- 1, Kamikodanaka 4-chome, Nakahara-ku, Kawasaki-shi,
Kanagawa 211-8588, Japan

☐ この欄に記載した者は、発明者でもある。

電話番号：044-754-3034

ファクシミリ番号：044-754-3563

加入電信番号：

国籍（国名）：日本国 JAPAN

住所（国名）：日本国 JAPAN

この欄に記載した者は、次の指定国についての出願人である：
☐ すべての指定国　☑ 米国を除くすべての指定国　☐ 米国のみ　☐ 追記欄に記載した指定国

第III欄　その他の出願人又は発明者

氏名（名称）及びあて名：(姓・名の順に記載；法人は公式の完全な名称を記載；あて名は郵便番号及び国名も記載)

岩下　英俊　IWASHITA　Hidetoshi
〒211－8588 日本国神奈川県川崎市中原区上小田中4丁目1番1号
富士通株式会社内
c/o FUJITSU LIMITED, 1-1, Kamikodanaka 4-chome, Nakahara-ku,
Kawasaki-shi, Kanagawa 211-8588, Japan

この欄に記載した者は次に該当する：
☐ 出願人のみである。
☑ 出願人及び発明者である。
☐ 発明者のみである。(ここにレ印を付したときは、以下に記入しないこと)

国籍（国名）：日本国 JAPAN

住所（国名）：日本国 JAPAN

この欄に記載した者は、次の指定国についての出願人である：
☐ すべての指定国　☐ 米国を除くすべての指定国　☑ 米国のみ　☐ 追記欄に記載した指定国

☐ その他の出願人又は発明者が続葉に記載されている。

第IV欄　代理人又は共通の代表者、通知のあて名

次に記載された者は、国際機関において出願人のために行動する：☑ 代理人　☐ 共通の代表者

氏名（名称）及びあて名：(姓・名の順に記載；法人は公式の完全な名称を記載；あて名は郵便番号及び国名も記載)

7409 弁理士 大菅 義之　OSUGA　Yoshiyuki
〒102－0084 日本国東京都千代田区二番町8番地20 二番町ビル3F
3rd Fl., Nibancho Bldg., 8-20 Nibancho, Chiyoda-ku,
Tokyo 102-0084, JAPAN

電話番号：03-3238-0031

ファクシミリ番号：03-3238-0034

加入電信番号：

☐ 通知のためのあて名：代理人又は共通の代表者が選任されておらず、上記枠内に特に通知が送付されるあて名を記載している場合は、レ印を付す

第Ⅴ欄　国の指定

規則 4.9(a)の規定に基づき次の指定を行う（*該当する□にレ印を付すこと；　少なくとも1つの□にレ印を付すこと*）。

広域特許

☐ ＡＰ　ＡＲＩＰＯ特許：ＧＨ ガーナ Ghana, ＧＭ ガンビア Gambia, ＫＥ ケニア Kenya, ＬＳ レソト Lesotho, ＭＷ マラウイ Malawi, ＳＤ スーダン Sudan, ＳＺ スワジランド Swaziland, ＵＧ ウガンダ Uganda, ＺＷ ジンバブエ Zimbabwe, 及びハラレプロトコルと特許協力条約の締約国である他の国

☐ ＥＡ　ユーラシア特許：ＡＭ アルメニア Armenia, ＡＺ アゼルバイジャン Azerbaijan, ＢＹ ベラルーシ Belarus, ＫＧ キルギス Kyrgyzstan, ＫＺ カザフスタン Kazakhstan, ＭＤ モルドヴァ Republic of Moldova, ＲＵ ロシア Russian Federation, ＴＪ タジキスタン Tajikistan, ＴＭ トルクメニスタン Turkmenistan, 及びユーラシア特許条約と特許協力条約の締約国である他の国

☐ ＥＰ　ヨーロッパ特許：ＡＴ オーストリア Austria, ＢＥ ベルギー Belgium, ＣＨ and ＬＩ スイス及びリヒテンシュタイン Switzerland and Liechtenstein, ＣＹ キプロス Cyprus, ＤＥ ドイツ Germany, ＤＫ デンマーク Denmark, ＥＳ スペイン Spain, ＦＩ フィンランド Finland, ＦＲ フランス France, ＧＢ 英国 United Kingdom, ＧＲ ギリシャ Greece, ＩＥ アイルランド Ireland, ＩＴ イタリア Italy, ＬＵ ルクセンブルグ Luxembourg, ＭＣ モナコ Monaco, ＮＬ オランダ Netherlands, ＰＴ ポルトガル Portugal, ＳＥ スウェーデン Sweden, 及びヨーロッパ特許条約と特許協力条約の締約国である他の国

☐ ＯＡ　ＯＡＰＩ特許：ＢＦ ブルキナ・ファソ Burkina Faso, ＢＪ ベナン Benin, ＣＦ 中央アフリカ Central African Republic, ＣＧ コンゴー Congo, ＣＩ コートジボアール Côte d'Ivoire, ＣＭ カメルーン Cameroon, ＧＡ ガボン Gabon, ＧＮ ギニア Guinea, ＭＬ マリ Mali, ＭＲ モーリタニア Mauritania, ＮＥ ニジェール Niger, ＳＮ セネガル Senegal, ＴＤ チャード Chad, ＴＧ トーゴー Togo, 及びアフリカ知的所有権機構のメンバー国と特許協力条約の締約国である他の国 *(他の種類の保護又は取扱いを求める場合には点線上に記載する)*

.....

国内特許 *(他の種類の保護又は取扱いを求める場合には点線上に記載する)*

☐ ＡＬ アルバニア Albania .....
☐ ＡＭ アルメニア Armenia .....
☐ ＡＴ オーストリア Austria .....
☐ ＡＵ オーストラリア Australia .....
☐ ＡＺ アゼルバイジャン Azerbaijan
☐ ＢＡ ボスニア・ヘルツェゴヴィナ Bosnia and Herzegovina .....
☐ ＢＢ バルバドス Barbados
☐ ＢＧ ブルガリア Bulgaria .....
☐ ＢＲ ブラジル Brazil .....
☐ ＢＹ ベラルーシ Belarus .....
☐ ＣＡ カナダ Canada
☐ ＣＨ and ＬＩ スイス及びリヒテンシュタイン Switzerland and Liechtenstein
☐ ＣＮ 中国 China .....
☐ ＣＵ キューバ Cuba .....
☐ ＣＺ チェッコ Czech Republic .....
☐ ＤＥ ドイツ Germany .....
☐ ＤＫ デンマーク Denmark .....
☐ ＥＥ エストニア Estonia .....
☐ ＥＳ スペイン Spain .....
☐ ＦＩ フィンランド Finland .....
☐ ＧＢ 英国 United Kingdom
☐ ＧＥ グルジア Georgia .....
☐ ＧＨ ガーナ Ghana .....
☐ ＧＭ ガンビア Gambia
☐ ＧＷ ギニア・ビサオ Guinea-Bissau .....
☐ ＨＲ クロアチア Croatia
☐ ＨＵ ハンガリー Hungary .....
☐ ＩＤ インドネシア Indonesia
☐ ＩＬ イスラエル Israel .....
☐ ＩＳ アイスランド Iceland
☑ ＪＰ 日本 Japan .....
☐ ＫＥ ケニア Kenya .....
☐ ＫＧ キルギス Kyrgyzstan .....
☐ ＫＲ 韓国 Republic of Korea .....
☐ ＫＺ カザフスタン Kazakhstan .....
☐ ＬＣ セント・ルシア Saint Lucia
☐ ＬＫ スリ・ランカ Sri Lanka
☐ ＬＲ リベリア Liberia
☐ ＬＳ レソト Lesotho .....
☐ ＬＴ リトアニア Lithuania
☐ ＬＵ ルクセンブルグ Luxembourg
☐ ＬＶ ラトヴィア Latvia
☐ ＭＤ モルドヴァ Republic of Moldova .....
☐ ＭＧ マダガスカル Madagascar .....
☐ ＭＫ マケドニア旧ユーゴースラヴィア共和国 The former Yugoslav Republic of Macedonia .....
☐ ＭＮ モンゴル Mongolia
☐ ＭＷ マラウイ Malawi .....
☐ ＭＸ メキシコ Mexico .....
☐ ＮＯ ノールウェー Norway
☐ ＮＺ ニュー・ジーランド New Zealand .....
☐ ＰＬ ポーランド Poland .....
☐ ＰＴ ポルトガル Portugal .....
☐ ＲＯ ルーマニア Romania
☐ ＲＵ ロシア Russian Federation .....
☐ ＳＤ スーダン Sudan
☐ ＳＥ スウェーデン Sweden
☐ ＳＧ シンガポール Singapore
☐ ＳＩ スロヴェニア Slovenia .....
☐ ＳＫ スロヴァキア Slovakia .....
☐ ＳＬ シエラ・レオーネ Sierra Leone
☐ ＴＪ タジキスタン Tajikistan .....
☐ ＴＭ トルクメニスタン Turkmenistan .....
☐ ＴＲ トルコ Turkey .....
☐ ＴＴ トリニダッド・トバゴ Trinidad and Tobago .....
☐ ＵＡ ウクライナ Ukraine .....
☐ ＵＧ ウガンダ Uganda .....
☑ ＵＳ 米国 United States of America .....
.....
☐ ＵＺ ウズベキスタン Uzbekistan .....
☐ ＶＮ ヴィエトナム Viet Nam .....
☐ ＹＵ ユーゴースラヴィア Yugoslavia .....
☐ ＺＷ ジンバブエ Zimbabwe .....

以下の□は、この様式の施行後に特許協力条約の締約国となった国を指定（国内特許のために）するためのものである

☐ .....
☐ .....
☐ .....
☐ .....
☐ .....

確認の指定の宣言：出願人は、上記の指定に加えて、規則 4.9(b)の規定に基づき、特許協力条約の下で認められる他の全ての国の指定を行う。ただし、この宣言から除く旨の表示を追記欄にした国は、指定から除かれる。出願人は、これらの追加される指定が確認を条件としていること、並びに優先日から15月が経過する前にその確認がなされない指定は、この期間の経過時に、出願人によって取り下げられたものとみなされることを宣言する。　*(指定の確認は、指定を特定する通知の提出と指定手数料及び確認手数料の納付からなる。この確認は、優先日から15月以内に受理官庁へ提出しなければならない。)*

第VI欄　優先権主張

☐ 他の優先権の主張（先の出願）が追記欄に記載されている

先の出願日 (日. 月. 年)	先の出願番号	先の出願		
		国内出願：国名	広域出願：*広域官庁名	国際出願：受理官庁名
(1)				
(2)				
(3)				

☐ 上記（ ）の番号の先の出願（ただし、本国際出願が提出される受理官庁に対して提出されたものに限る）のうち、次の（ ）の番号のものについては、出願書類の認証謄本を作成し国際事務局へ送付することを、受理官庁（日本国特許庁の長官）に対して請求している。　：

*先の出願が、ARIPOの特許出願である場合には、その先の出願を行った工業所有権の保護のためのパリ条約同盟国の少なくとも1ヶ国を追記欄に表示しなければならない（規則4．10(b)(ii)）。追記欄を参照。

第VII欄　国際調査機関

国際調査機関（ISA）の選択

ISA／JP

先の調査結果の利用請求；当該調査の照会（先の調査が、国際調査機関によって既に実施又は請求されている場合）

出願日（日. 月. 年）　出願番号　国名（又は広域官庁）

第VIII欄　照合欄：出願の言語

この国際出願の用紙の枚数は次のとおりである。

願書	3	枚
明細書（配列表を除く）	36	枚
請求の範囲	3	枚
要約書	1	枚
図面	30	枚
明細書の配列表	0	枚
合計	73	枚

この国際出願には、以下にチェックした書類が添付されている。

1. ☑ 手数料計算用紙
   ☑ 納付する手数料に相当する特許印紙を貼付した書面
   ☑ 国際事務局の口座への振込みを証明する書面
2. ☐ 別個の記名押印された委任状
3. ☑ 包括委任状の写し
4. ☐ 記名押印（署名）の説明書
5. ☐ 優先権書類（上記第VI欄の（ ）の番号を記載する）：
6. ☐ 国際出願の翻訳文（翻訳に使用した言語名を記載する）：
7. ☐ 寄託した微生物又は他の生物材料に関する書面
8. ☐ ヌクレオチド又はアミノ酸配列表（フレキシブルディスク）
9. ☐ その他（書類名を詳細に記載する）：

要約書とともに提示する図面：　図5

本国際出願の使用言語名：　日本語

第IX欄　提出者の記名押印

各人の氏名（名称）を記載し、その次に押印する。

大菅義之

受理官庁記入欄

1．国際出願として提出された書類の実際の受理の日　29.01.99

2．図面
☐ 受理された
☐ 不足図面がある

3．国際出願として提出された書類を補完する書類又は図面であって
その後期間内に提出されたものの実際の受理の日（訂正日）

4．特許協力条約第11条(2)に基づく必要な補完の期間内の受理の日

5．出願人により特定された国際調査機関　ISA／JP

6．☐ 調査手数料未払いにつき、国際調査機関に調査用写しを送付していない

国際事務局記入欄

記録原本の受理の日

この用紙は、国際出願の一部を構成せず、国際出願の用紙の枚数に算入しない。

PCT/JP00/00381

P C T

手数料計算用紙

願書附属書

受理官庁記入欄

国際出願番号

受理官庁の日付印

出願人又は代理人の書類記号

9805482

出願人

富士通株式会社

所定の手数料の計算

1．及び2．特許協力条約に基づく国際出願等に関する法律（国内法）第18条第1項第1号の規定による手数料 *(注1)*
（送付手数料［T］及び調査手数料［S］の合計）

95,000.- 円 | T+S | 95,000-

3．国際手数料 *(注2)*

基本手数料

国際出願に含まれる用紙の枚数 73 枚

最初の30枚まで ..... 62,800.- 円 | b1 | 62,800-

43 （30枚を越える用紙の枚数） × 1,450 （用紙1枚の手数料） = 62,350.- 円 | b2 | 62,350-

b1及びb2に記入した金額を加算し、合計額をBに記入 125,150.- 円 | B | 125,150-

指定手数料

国際出願に含まれる指定数 *(注3)* 2

2 （支払うべき指定手数料の数 *(上限は11)* *(注4)*） × 14,500 （1指定当たりの手数料（円）） = 29,000.- 円 | D | 29,000-

B及びDに記入した金額を加算し、合計額をTに記入 ..... 154,150.- 円 | T | 154,150-

4．納付すべき手数料の合計

T+S及びTに記入した金額を加算し、合計額を合計に記入 249,150.- 円

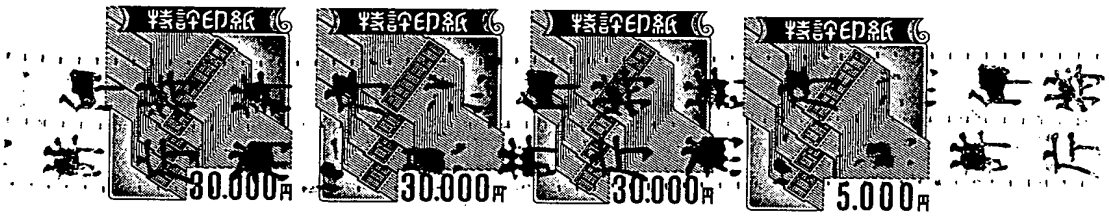
合計 | 249,150-

(注1) 送付手数料及び調査手数料については、合計金額を特許印紙をもって納付しなければならない。

(注2) 国際手数料については、受理官庁である日本国特許庁の長官が告示する国際事務局の口座への振込みを証明する書面を提出することにより納付しなければならない。

(注3) 願書第V欄でレ印を付した口の数。

(注4) 指定数を記入する。 ただし、11指定以上は一律11とする。



送付手数料・調査手数料　95,000 円

PCT/JP99/00381

振込金受取書

電信扱 | 文書扱

さくら銀行

（兼手数料受取書）

さくら銀行をご利用いただきありがとうございました。

●次回より早くて便利なATMをご利用いただきますと、手数料がお得になります
●午後2時以降は、窓口が大変混雑いたしますので、お振込は出来るだけ早めにご依頼ください。

11年 1月29日

東京三菱 内幸町 店

金額 ¥154150

普通 2. 当座 4. 貯蓄 9. ①

口座番号 0473286

お受取人

フリカナ

おなまえ WIPO - PCT GENEVA 様

おてんわ － －

ご依頼人

フリカナ

おなまえ オオスガナイガイコクトッキョジムショ
大菅内外国特許事務所 様

おところ オオ 大 スガ 菅 ヨシ 義 ユキ 之

〒102 東京都千代田区二番町8-20 二番町ビル3階
ご連絡先お電話3238-0031・FAX3238-0034

さくら銀行 11.-1.29 83 吉田

¥420 ¥315

¥525 ¥420

¥735 ¥630

株式会社さくら銀行

振込依頼書に記載相違等の不備があった場合には、照会等のため振込が遅延したり、振込ができないことがあります。
通信機器、回線の障害または[illegible]の遅延等やむを得ない事由によって、振込が遅延することがありますのでご了承ください。
この振込金受取書は、[illegible]などに必要となりますので、ご依頼人が大切に保管してください。
文書扱は、お受取様への到着に日時がかかりますからご承知おきください。

基本手数料	125,150 円
指定手数料	29,000 円
合計	154,150 円

01.99

PCT/JP96/00381

包　括　委　任　状

平成 8 年 8 月 27日

私儀　弁理士　　大菅義之　氏
を代理人と定めて下記の権限を委任します。

1. 特許協力条約に基づくすべての国際出願に関する一切の件

2. 上記出願又は指定国の指定を取り下げる件

3. 上記出願に対する国際予備審査の請求に関する一切の件並びに
選択国の選択を取り下げる件

あて名　〒211 日本国神奈川県川崎市中原区上小田中４丁目１番１号
名　称　富　士　通　株　式　会　社
代表取締役社長　　関澤　義

明　細　書

コンパイラ装置、コンパイル方法、およびそのためのプログラムを格納した記憶媒体

技術分野

本発明は、ある言語で記述された入力プログラムの少なくとも一部をコンパイルしてその言語と同一の言語またはその言語と異なる他の言語で記述されたプログラムを出力するコンパイラに係わる。特に、出力プログラムがFortranまたはＣ言語などであるプリプロセッサ方式のコンパイラに係わる。

背景技術

コンピュータの動作を記述するためのプログラミング言語は、使用目的や適用分野などに応じて様々な拡張言語または改良言語が開発されている。たとえば、Fortran をベースとする拡張言語としては、ＨＰＦが広く知られている。また、Ｃ言語をベースとする拡張言語としては、C++、C9x、ＤＰＣＥなどが知られている。

ところが、これらの拡張言語で記述されたプログラム等をアセンブラまたは機械語に翻訳する際には、いったんベース言語に変換することが有利なことが多い。このため、これらの拡張言語を扱うコンパイラは、一般に、出力の言語がその拡張言語のベース言語であるプリプロセッサ方式で実現されることが多い。たとえば、ＨＰＦで記述されたプログラムをFortranで記述されたプログラムに変換して出力するコンパイラが開発されている。

関数 SUM_SCATTER は、下記の形式で呼び出される。

SUM_SCATTER(ARRAY, BASE, INDX1, INDX2, ... , INDXn, MASK)

ここで、「ARRAY 」は、任意の数値型であり、且つ任意の次元数を持つ配列である。数値型は、処理系が許す任意のバイト数の整数型、実数型、複素数型などである。「BASE」は、「ARRAY 」と同じ型であり、任意の次元数を持つ配列である。「INDX1, INDX2, ... , INDXn」は、スカラであるか「BASE」と同じ次元数を持つ配列であり、「ｎ」は「ARRAY 」と同じ次元数である。「MASK」は、スカラであるか、「ARRAY 」と同じ次元数を持つ配列であるか、あるいは省略される。

したがって、たとえば、もし、処理系が１０種類の数値型を持ち、１次元～７次元までの配列をサポートすると仮定すると、この関数の自由度は以下のように計算できる。

「ARRAY」が１次元のときは、「ARRAY」の自由度が１０、「BASE」の自由度が７、「ｎ＝１」なので「INDX1, INDX2, ... , INDXn」の自由度が２、さらに「MASK」の自由度が３である。従って、この場合、組合せの数は、１０×７×２×３＝４２０となる。同様に、「ARRAY」が２次元～７次元のときの組合せの数は以下の通りである。

２次元：$10\times7\times2^{2}\times3=840$

３次元：$10\times7\times2^{3}\times3=1680$

.

.

.

７次元：１０×７×２^{7}　　×３＝２６８８０

よって、合計の組合せ数は、５３３４０通りになる。さらに、ＨＰＦでは、一般に、配列は複数のプロセッサに分割して配置されるので、分割配置の種類（block, cyclic など）や分割次元の違い、および多次元分割などを考慮すると、その組合せは膨大なものとなる。

したがって、抽象度の高い組込み手続の変換を、通常のライブラリを用いて実現しようとすると、以下の問題が生じる。

(1) すべての組合せに対応しようとすると、ライブラリが巨大になり、結果としてそのライブラリを記憶するための記憶媒体（メモリ容量）も大きくなってしまう。

(2) 複数のケースに対応できるような入口名（総称的な名前、あるいは抽象的な名前）を使ったとしても、結局はライブラリの中でケース分けをする必要があるので、上記(1) の問題は解決されない。

(3) 複数のケースに対応できるような入口名（総称的な名前、あるいは抽象的な名前）を使い、且つ様々な型や次元数でライブラリを作る方法も考えられるが、実際には以下の問題を伴う。

(a) 次元数の違いに対応するためには、ループのネスト数を可変にできればよいが、Fortran やＣ言語ではそのような記述はできない。また、再帰呼出しで記述すると、実行速度が低下する。

(b) 型（たとえば、４バイトの整数型や８バイトの実数型）を共通化することは困難である。たとえば、プログラム中の式の中に整数型の数値と実数型の数値が混在していた場合、これらの型は実行コードが互いに異なるので、型ごとに処理を分ける必要がある。このため、命令列中に判断分岐が多数登場す

ることになり、実行速度が低下する。

(c) 複数のプロセッサに分割された配列を引数としてサポートする言語においては、単純な分割（例えば、最終次元での均等なblock分割）に対しても一般的な分割（すべての次元において任意の分割方法で分割できる方法を想定する）と同じ扱いになるので、実行速度が低下してしまう。

このように、抽象度の高い組込み手続をプリプロセッサ方式で変換しようとすると、ライブラリを記憶するための記憶媒体が巨大になってしまった。あるいは、記憶媒体の巨大化を回避しようとすると、コンパイル能力が低下していた。

この問題を回避する方法の１つとして、組込み手続のすべての部分についてライブラリを用意しておくのではなく、組込み手続の呼出しパラメータに係わる部分のみをコンパイル時に生成（具体化）する方法が考えられる。例えば、インライン展開技術を用いてソースプログラム中の手続き呼出を変換する手法が考えられる。

ところが、プリプロセッサ方式のコンパイラにおいては、手続き呼出の登場場所によっては、インライン展開が出来なかったり、困難であったりすることがある。以下、問題点を説明する。

(1) 通常、宣言部において実行文を展開することは出来ない。このため、宣言部に関数呼出等が存在する場合は、それをインライン展開することはできない。もし、宣言部に登場する関数呼出等をそのままインライン展開すると、得られたプログラムをベース言語のコンパイラで翻訳したときにエラーが発生することになる。たとえば、図１Ａに示す入力プログラムの宣言部をFortranプログラムにコンパイルすると、図１Ｂに示すコードが得られる。また、図２Ａ

に示す入力プログラムの宣言部をCプログラムにコンパイルすると、図２Ｂに示すコードが得られる。これらの例では、宣言部において実行文（図１の例では、関数ABS 、図２の例では、関数strlen(s) ）が展開されているので、コンパイル結果として得られるプログラムをそれぞれFortranコンパイラまたはCコンパイラで翻訳すると、エラーが発生してしまう。

(2) 実行部においても、文法上の制約により、インライン展開が出来ない場合がある。たとえば、Fortran 90では、「where文の中では代入文以外の実行文を記述してはいなない」という制約があるので、where 文の中に登場する関数をインライン展開できない場合が生じる。

(3) 名前の衝突の回避が困難な場合がある。たとえば、展開すべき手続内で組込み関数SUMが使用されている場合、もし、呼び出される側で変数名SUMが使われており、この変数が呼出の実引数として現れるとすると、同一の名前が重複するという問題を回避することは困難である。

(4) 実行文等を展開する場合、それを単にその場所に展開するだけでなく、文脈に応じて周辺のコードも変換しなければならないことがある。たとえば、図３Ａに示すように、入力プログラムの中のELSEIF文の条件節に関数呼出（図３の例では、関数ABS）が現れる場合には、IF文の入れ子構造を変換する必要が生じる。このため、図３Ｂに示すように、インライン展開が複雑になる。また、入力プログラムの中のDO WHILE文の条件節に呼出し関数が現れる場合は、その関数をループの直前およびループ内の双方で展開しなければならず、この場合もインライン展開が複雑になる。一方、Ｃ言語では、たとえば、for文の括弧内の２番目または３番目の式を展開する場合には、同様にインライン展開が複雑になる。

このように、プログラム等をコンパイルする際、インライン展開が可能であれば、そのコンパイルに必要な情報を格納するための記憶領域を小さくすることができるなどのメリットが得られるが、オブジェクト言語が処理手続を記述できる場所について制約がある言語であった場合には、インライン展開が出来なかったり、あるいは困難だったりする。すなわち、オブジェクト言語が機械語やアセンブラのように実行コードの出現場所についての制約が少ないコンパイラにおいては、インライン展開をするにあたっての問題も少ないが、オブジェクト言語がFortran やC言語などのように実行コードの出現場所についての制約が多いプリプロセッサ方式のコンパイラにおいては、インライン展開をするにあたって上述のような問題が発生しやすい。

本発明の課題は、実行速度が高速で、且つコンパイルに必要な情報を格納するための記憶領域が小さくなるプリプロセッサ方式のコンパイラを提供することである。

発明の開示

本発明のコンパイラ装置は、入力された第１のプログラムをコンパイルして第２のプログラムを出力する構成を前提とし、第１のプログラムの一部を手続呼出に変換して上記第２のプログラムを作成する変換ユニットと、上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成する生成ユニットと、上記第２のプログラムおよび上記生成手段により生成されたコードを出力する出力ユニットとを有する。

第１のプログラムをコンパイルすることにより第２のプログラムを作成する際、手続の定義を記述したコードは第２のプログラムの外に生成され、第２の

プログラム中にはそのコードを呼び出すための手続呼出が記述される。すなわち、第２のプログラム中には実行コードは展開されない。したがって、オブジェクト言語が処理手続を記述できる場所について制約がある言語であった場合においても、実行されたときにエラーの発生を伴わない第２のプログラムが得られる。また、第２のプログラムの記述が複雑になることはなく、インライン展開を用いる場合と比べ、第２のプログラムをベース言語コンパイラで翻訳する速度の向上が期待される。さらに、上記装置においては、入力プログラム中の変換すべき対象のすべての組合せに対してコンパイルに必要な情報を用意しておく必要がないので、そのためのメモリ容量が小さくなる。

図面の簡単な説明

図１Ａおよび図１Ｂは、従来のコンパイラによってインライン展開できない例（その１）である。

図２Ａおよび図２Ｂは、従来のコンパイラによってインライン展開できない例（その２）である。

図３Ａおよび図３Ｂは、従来のコンパイラによってインライン展開が複雑になる例である。

図４は、本実施例のコンパイラの概略動作の一例を説明する図である。

図５は、本実施例のコンパイラの機能ブロック図である。

図６は、図５に示したコンパイラの動作を説明するためのフローチャートである。

図７Ａは、図５に示したコンパイラへの入力コードの例である。

図７Ｂは、図５に示したコンパイラからの出力コードの例である。

図８は、関数 SUM を展開するためのテンプレートの例である。

図９Ａおよび図９Ｂは、テンプレートを用いて関数 SUM を展開した結果を示す図である。

図１０は、本発明の他の形態のコンパイラの機能ブロック図である。

図１１は、図１０に示したコンパイラの動作を説明するためのフローチャートである。

図１２Ａは、図１０に示したコンパイラへの入力コードの例である。

図１２Ｂは、図１０に示したコンパイラからの出力コードの例である。

図１３Ａ～図１３Ｄは、重複判定において使用する管理テーブルを説明する図である。

図１４は、関数 SUM を展開した結果を示す図である。

図１５は、抽象度の高い表現で作成したテンプレートの例である。

図１６は、図１５に示すテンプレートを用いて変換すべき対象を展開する際に比較されるパラメータを示す図である。

図１７は、図１５に示すテンプレートを用いた場合のコンパイラの出力を示す図である。

図１８は、本発明の更に他の形態のコンパイラの機能ブロック図である。

図１９は、図１８に示したコンパイラの動作を説明するためのフローチャートである。

図２０は、図１８に示したコンパイラへの入力コードの例である。

図２１は、図１８に示したコンパイラからの出力コードの例である。

図２２は、本発明の更に他の形態のコンパイラの機能ブロック図である。

図２３は、図２２に示したコンパイラの動作を説明するためのフローチャー

トである。

図２４は、図２２に示したコンパイラへの入力コードの例である。

図２５は、図２２に示したコンパイラからの出力コードの例である。

図２６Ａは、入力コードの例である。

図２６Ｂは、図２６Ａに示す入力コードに対応する出力の例である。

図２７Ａ～図２７Ｄ、図２８Ａ、および図２８Ｂは、テンプレートの例である。

図２９は、本発明に係わるソフトウェアプログラムを実行するコンピュータのブロック図である。

図３０は、本発明に係わるソフトウェアプログラムの提供方法を説明する図である。

発明を実施するための最良の形態

本発明は、入力されたプログラムの少なくとも一部をコンパイルするプリプロセッサ方式のコンパイラに係わる。プリプロセッサ方式のコンパイラ（「プリコンパイラ」とも呼ばれている）は、ソースプログラムを読み込み、そのソースプログラムの少なくとも一部を変更してそのソースプログラムのコンパイルのための準備を行うプログラムである。以下の実施例では、Fortran およびＣ言語を採り上げ、ソースプログラムの中の呼出（関数、サブルーチン、ライブラリ等の組込み手続を呼び出すための呼出）などをオブジェクトコードに変換する機能を中心に説明する。なお、本実施例のコンパイラは、ソース言語とオブジェクト言語とが互いに異なっている場合、およびソース言語とオブジェクト言語とが互いに同じ場合の双方に適用可能である。

本実施例のコンパイラの概略動作の一例を図４を参照しながら説明する。ここでは、ソースプログラム１が、たとえば、Fortran の拡張言語の１つであるＨＰＦで記述されており、その中に「Ａ＊＊Ｎ」が含まれているものとする。「Ａ＊＊Ｎ」は、「ＡのＮ乗」を表す。また、オブジェクト言語は、例えば、Fortran であるものとする。この場合、コンパイラ２は、たとえば、ソースプログラム１の中の「Ａ＊＊Ｎ」を「ＰＯＷ（Ａ，Ｎ）」に変換することによりオブジェクトプログラム３を生成すると共に、「ＰＯＷ（Ａ，Ｎ）」により呼び出されるべき手続を記述するコード（オンラインコード）４を出力する。この手続の定義は、たとえば、「Ｎが所定値よりも小さいときには掛け算を実行し、Ｎが所定値以上のときには巾乗演算を実行する。」である。また、コード４は、この手続をオンライン展開することにより得られる。なお、「オンライン展開」とは、あるプログラム中に登場する手続等をそのプログラムの外でサブプログラムとして展開することを意味する。また、オンライン展開により得られるコードを「オンラインコード」と呼ぶことにする。

なお、コンパイラ２は、必要に応じて、オブジェクトプログラム３の手続呼出を用いてオンラインコード４を呼び出すための記述（たとえば、インタフェース宣言）をオブジェクトプログラム３に加える。

このように、コンパイラ２は、ソースプログラム１の一部を手続呼出に変換することによりオブジェクトプログラム３を生成する際、その手続呼出により呼び出されるべきコードをオンラインコード４としてオブジェクトプログラム３の外に作成する。すなわち、手続呼出により呼び出されるべきコードは、オブジェクトプログラム３の中では展開されない。従って、オブジェクト言語が手続を記述できる場所について制約がある言語であったとしても、オブジェク

トプログラム３が実行されたときに文法エラー等が発生することはない。

図５は、本実施例のコンパイラの機能ブロック図である。ここでは、プログラム毎にソースコードが与えられることを前提とする。

ソースプログラムは、構文解析部２１により分解・解析された後にコンパイラ１０に入力される。構文解析部２１は、入力コードを中間コードに変換する機能を備え、既存のプログラムをそのまま流用できる。

コンパイラ１０は、検出部１１、変換部１２、および展開部１３を備える。検出部１１は、変換すべき対象を検出する。具体的には、検出部１１は、ソースプログラムから予め決められた特定のパターン（たとえば、手続呼出など）を検出する。変換部１２は、検出部１１により検出された変換すべき対象に対応する手続名を決定し、その変換すべき対象をその手続名に置き換える。そして、コンパイラ１０は、変換部１２によりその一部が変換されたソースプログラムをオブジェクトプログラムとして出力する。展開部１３は、検出部１１により検出された変換すべき対象に対応するオンラインコード（手続コードまたは実行コード）を生成する。なお、このオンラインコードは、オブジェクトプログラムが実行されたときに、変換部１２により決定された手続名に対応付けられて呼び出される。そして、コンパイラ１０は、その展開部１３により生成されたオンラインコードを出力する。

コード生成部２２は、コンパイラ１０から出力されたオブジェクトプログラムおよびオンラインコードをアセンブラまたは機械語に翻訳し、それらをそれぞれ出力する。コード生成部２２は、既存のプログラムをそのまま流用でき、最適化処理を行うこともできる。

図６は、図５に示したコンパイラ１０の動作を説明するためのフローチャー

トである。ここでは、コンパイラ１０に入力されるソースプログラムをプラグラムＰ、コンパイラ１０によりコンパイルされたオブジェクトプログラムをプログラムＰ’と呼ぶことにする。なお、この処理は、１つのソースプログラムが入力される毎に実行される。

ステップＳ１では、プログラムＰ内で変換すべき対象Ｃ1 ,....,Ｃn を検出する。変換すべき対象Ｃ1 ,....,Ｃn は、たとえば、関数、サブルーチン、ライブラリの呼出である。

ステップＳ２～Ｓ５の処理は、ステップＳ１において検出された各変換すべき対象Ｃ1 ,....,Ｃn について実行される。ステップＳ２では、変換すべき対象Ｃi の特徴Ａi を抽出する。「特徴」とは、例えば、関数名などの呼び出すべき手続の名前、引数の次元、数値型などである。ステップＳ３では、変換すべき対象Ｃi に対して、プログラムＰ内でユニークな手続名ｆi を生成する。すなわち、各変換すべき対象Ｃ1,....,Ｃn に対して、互いに異なる手続名を割り当てる。

ステップＳ４では、変換すべき対象Ｃi を手続ｆi の呼出に置換する。ここで、「手続ｆi の呼出」とは、当該プログラムが実行されたときに、手続ｆi を呼び出すための呼出である。ステップＳ５では、特徴Ａi に対応する手続コードＳi を生成する。そして、この手続コードＳi の手続名としてｆi を指定する。これにより、手続コードＳi は、当該プログラムが実行されたときに、手続ｆi の呼出により呼び出されることになる。

上記ステップＳ２～Ｓ５の処理をすべての変換すべき対象Ｃ1 ,....,Ｃn について実行すると、プログラムＰにおいて各変換すべき対象Ｃ1 ,....,Ｃn がそれぞれ手続ｆ1 ,....,ｆn の呼出に置換され、また、各手続ｆ1 ,....,ｆn

の呼出にそれぞれ対応して手続コードＳ１，....，Ｓｎ が生成される。そして、ステップＳ６において、変換すべき対象Ｃｉ を手続ｆｉ の呼出に置き換えることにより得られるプログラムＰ’、及び生成された手続コードＳ１，....，Ｓｎ が出力される。なお、プログラムＰ’および手続コードＳ１，....，Ｓｎ は、互いに同じファイルに出力されてもよいし、互いに異なるファイルに出力されてもよい。また、ソースコードが複数のプログラム単位を含む場合には、各プログラム単位に対して上記処理が繰り返し実行される。

次に、具体的な実施例を説明する。以下では、コンパイラ１０に対して図７Ａに示すソースプログラムが入力された場合を想定し、関数 SUM の呼出を変換すべき対象として説明する。なお、関数 SUM は、引数により指定された配列の要素の総和を返す関数であるものとする。また、ソースプログラムは、中間コードに変換されていてもよい。

ソースプログラムが入力されると、コンパイラ１０は、そのソースプログラムをスキャンすることにより関数 SUM の呼出を検出する。図７Ａに示す例においては、関数 SUM は、４行目の「SUM(A)」および５行目の「SUM(N(51:100))」により呼ばれていることが検出される。

続いて、コンパイラ１０は、「SUM(A)」をユニークな名前が割り当てられた手続の呼出に変換する。この実施例では、新たに生成される名前は、変換すべき対象の手続名と、ソースプログラムの名前と、そのソースプログラムの中で登場した順番との組合せにより決定される。この場合、図７Ａに示す例では、「SUM(A)」はプログラム SAMPLE の中で最初に登場した変換対象なので、新たな手続名として「SUM_SAMPLE_1」が割り当てられる。なお、コンパイラ１０が新たに生成する名前とユーザにより定義される変数名などとの重複を回避する

ためには、コンパイラ１０は、ユーザが使用できない名前を選択するようにすればよい。たとえば、「xxx_」で始まる名前を予め予約しておくことによりユーザの使用を制限する方法や、「＃」や「＄」などのユーザが滅多に使用しない文字を含む名前を生成するようにする方法が考えられる。

この後、コンパイラ１０は、ソースプログラムの４行目を以下のように変換する。このとき、「SUM_SAMPLE_1」の引数は、「SUM(A)」の引数をそのまま使用する。

B=SUM(A) → B=SUM_SAMPLE_1(A)

続いて、コンパイラ１０は、引数Ａを解析することにより、「SUM(A)」の特徴を抽出する。これにより、以下の情報が得られる。

arg-type ＝ ＲＥＡＬ

m＝３

lb(1) ＝１

ub(1) ＝１０

lb(2) ＝１

ub(2) ＝２０

lb(3) ＝１

ub(3) ＝３０

ここで、「arg-type」は引数の型、「m」は引数の次元数、「lb(i) 」および「ub(i) 」はそれぞれ引数の第ｉ次元の下限値および上限値である。

コンパイラ１０は、「SUM_SAMPLE_1」の手続コードを生成する際、テンプレートを利用する。テンプレートは、予め決められた変換すべき対象を展開するためのひな形であり、データベース等に格納されている。関数 SUM を展開する

ためのテンプレートの例を図８に示す。

テンプレートは、抽象的な表現を用いて作成されており、変換すべき対象の特徴に応じてそのテンプレートをカスタマイズすることにより対応する手続コードが生成される。また、１つの呼出に対して複数のテンプレートを設けてもよい。例えば、重要な呼出に対しては複数種類のテンプレートを設けておき、その呼出の特徴に応じてそれら複数のテンプレートの中から適切な１つを選択して利用するようにしてもよい。さらに、手続の引数などをパラメータとして具体的なソースコードまたはオブジェクトコードを生成するような処理プログラムとしてテンプレートを作成することも可能である。

コンパイラ１０は、図８に示すテンプレートを利用し、引数Ａを解析することにより得られている情報に従って「SUM_SAMPLE_1(A) 」についての手続コードを生成する。このようにして生成された手続コードの例を図９Ａに示す。

手続コードは、具体的には、以下のようにして作成される。すなわち、引数Ａの「型」が「REAL」であるので、図８に示すテンプレートにおいて、「arg-type」として「REAL」が書き込まれる。また、引数の各次元の下限値および上限値として「１」「１０」「１」「２０」「１」「３０」が書き込まれる。さらに、このコードが「SUM_SAMPLE_1(A) 」により呼び出されるようにするために、手続名として「SUM_SAMPLE_1」が書き込まれる。

「SUM(N(51:100))」についての処理は、基本的に、「SUM(A)」についての処理と同じである。すなわち、コンパイラ１０は、まず「SUM(N(51:100))」をユニークな名前が割り当てられた手続の呼出に変換する。図７Ａに示す例では、「SUM(N(51:100))」はプログラム SAMPLE の中で２番目に登場した関数 SUM なので、新たな手続名として「SUM_SAMPLE_2」が割り当てられる。この後、コン

パイラ１０は、ソースプログラムの５行目を以下のように変換する。

WRITE(*,*) SUM(N(51:100)) → WRITE(*,*) SUM_SAMPLE_2(N(51:100))

続いて、コンパイラ１０は、引数を解析して「SUM(N(51:100))」の特徴を抽出する。これにより、以下の情報が得られる。

arg-type ＝ ＩＮＴＥＧＥＲ

m＝１

lb(1) ＝５１

ub(1) ＝１００

この後、コンパイラ１０は、図８に示したテンプレートを利用して「SUM_SAMPLE_2(N(51:100))」についての手続コードを生成する。生成された手続コードの例を図９Ｂに示す。

図７Ｂは、コンパイラ１０を用いて図７Ａに示したソースプログラムをコンパイルした結果を示す図であり、図６に示したフローチャートにおけるプログラムＰ’に相当する。図７Ａおよび図７Ｂに示すように、ソースプログラム中に複数の変換すべき対象（関数 SUM を呼び出すための呼出）が存在する場合、コンパイラ１０は、それらに対して互いに異なる手続名を割り当てることにより、それらの変換対象を互いに識別可能な手続呼出に置き換える。この変換処理により得られたコード（オブジェクトプログラム）は、図７Ｂに示すようなプログラムコードイメージか、対応するオブジェクトコードとしてファイルに出力される。また、コンパイラ１０は、各手続呼出に対して、手続コードを生成して出力する。ここで、コンパイラ１０は、複数の手続コードを一括して出力してもよいし、生成するごとに、順次、出力するようにうしてもよい。

このように、コンパイラ１０は、図７Ａに示すソースプログラムが入力され

ると、図７Ｂに示すオブジェクトコード、および図９Ａおよび図９Ｂに示す手続コードを出力する。ここで、手続コードは、オブジェクトコード内では展開されない。したがって、コンパイラ１０のオブジェクト言語が手続コードを記述できる場所について制約を持っていたとしても、コンパイル結果がその制約に違反することはない。

従来、たとえば、FORTRAN やＣ言語などでは、プログラムの宣言部において実行文を展開することができないので、宣言部に組込み手続等の呼出が記述されていた場合には、従来のコンパイラではその宣言部を正しくコンパイルすることはできなかった。ところが、本実施例のコンパイラによれば、組込み手続等の呼出はオブジェクト言語における手続呼出に置き換えられるだけなので、そのような問題は生じない。

上記図５～図９を参照しながら説明した実施例では、ソースプログラム中の変換すべき対象が互いに同一ではない場合を採り上げた。すなわち、図７Ａに示した例では、２つの変換すべき対象は、呼び出すべき関数は互いに同じであったが、その引数は互いに異なっていた。

ところが、ソースプログラム中に複数の互いに同一の変換すべき対象が存在することがある。この場合、もし、図５に示したコンパイラ１０を用いてそのソースプログラムをコンパイルすると、複数の互いに同一の手続コード（オンラインコード）を生成することになり、コンパイラの総出力コード量が必要以上に多くなってしまう。以下、この問題を解決したコンパイラについて説明する。

図１０は、本発明の他の形態のコンパイラの機能ブロック図である。図１０において使用する符号のうち、図５において先に使用しているものは同じ機能

を持つユニットを表す。すなわち、コンパイラ３０において、検出部１１、変換部１２、展開部１３は、基本的に図５に示したコンパイラ１０のものと同じユニットである。

コンパイラ３０は、重複判定部３１を備えることを特徴とする。重複判定部３１は、検出部１１によりソースプログラムにおいて検出された複数の変換すべき対象のうち、互いに同じものが存在するか否かを調べ、その結果を変換部１２および展開部１３に通知する。互いに同じ変換すべき対象が存在しなければ、変換部１２および展開部１３は、図５～図９を参照しながら説明した動作を実行する。一方、互いに同じ変換すべき対象が存在する場合は、変換部１２は、それらをそれぞれ互いに同じ呼出に変換し、展開部１３はそれらの呼出により共有的に呼び出される手続コード（オンラインコード）を１つだけ生成する。

図１１は、図１０に示したコンパイラの動作を説明するためのフローチャートである。

ステップＳ１１では、まず、重複判定部３１が使用するメモリ領域Ｒ（管理テーブル）をクリアする。なお、メモリ領域Ｒには、ソースプログラム中に存在する変換すべき対象の特徴が登録され、さらに、各特徴に対してユニークに割り当てられる手続名が登録される。ステップＳ１２は、図６のステップＳ１と同様に、プログラムＰ内で変換すべき対象Ｃ1 ,....., Ｃn を検出する。

ステップＳ１３～Ｓ１９の処理は、ステップＳ１２において検出された各変換すべき対象Ｃ1 ,....., Ｃn について実行される。ステップＳ１３では、図６のステップＳ２と同様に、変換すべき対象Ｃi の特徴Ａi を抽出する。ステップＳ１４では、特徴Ａi が、先にステップＳ１４～Ｓ１９の処理が終了してい

る変換すべき対象C1 ,....., Ci-1 の特徴の中のいずれかと同じであるか否かを調べる。特徴Ai が、先の処理がなされた変換すべき対象C1 ,....., Ci-1 の特徴の中のいずれかと同じであればステップS19へ進み、いずれにも一致しなければステップS15へ進む。

ステップS15では、メモリ領域Rをサーチし、変換すべき対象Ci に対してプログラムP内でユニークな手続名fi を生成する。ステップS16では、図6のステップS4と同様に、変換すべき対象Ci を手続fi の呼出に置換する。ステップS17では、図6のステップS5と同様に、特徴Ai に対応する手続コードSi を生成する。この手続コードSi の手続名としてfi が指定される。さらに、ステップS18において、変換すべき対象Ci の特徴Ai および対応する手続コードSi を呼び出すための手続名としてのfi がメモリ領域Rに登録される。

一方、ステップS19では、変換すべき対象Ci を、特徴Aj に対応する手続コードSj を呼び出すための手続fj の呼出に置換する。なお、ステップS19が実行された場合は、新たな手続コードが生成されず、また、メモリ領域Rへの新たな登録も行われない。

上記ステップS13～S19の処理を全ての変換すべき対象C1 ,....., Cn について実行すると、プログラムPにおいて各変換すべき対象C1 ,....., Cn が手続f1 ,....., fm (m≦n) の呼出に置換され、また、各手続f1 ,....., fm の呼出にそれぞれ対応して手続コードS1 ,....., Sm が生成される。そして、ステップS20において、変換すべき対象を手続の呼出に置き換えることにより得られるプログラムP'、および生成された手続コードが出力される。なお、プログラムP'および手続コードS1 ,....., Sm は、互いに同じファイ

ルに出力されてもよいし、互いに異なるファイルに出力されてもよい。

このように、コンパイラ３０によれば、あるソースプログラム内に存在する互いに重複した変換すべき対象に対して、重複して手続コードを生成することなく、必要最小限の手続コードを生成するので、コンパイラの総出力コード量が減少する。

次に、具体的な実施例を説明する。以下では、コンパイラ３０に対して図１２Ａに示すソースプログラムが入力された場合を想定し、関数 SUM の呼出を変換対象として説明する。なお、コンパイラ３０は、図８に示したテンプレートを利用するものとする。

ソースプログラムが入力されると、コンパイラ３０は、そのソースプログラムをスキャンすることによって関数 SUM の呼出を検出する。図１２Ａに示す例においては、関数 SUM は、４行目の「SUM(A)」および「SUM(A2) 」、５行目の「SUM(N(51:100))」、および６行目の「SUM(M(51:200))」により呼ばれていることが検出される。

コンパイラ３０は、まず、「SUM(A)」についてその特徴を抽出し、その特徴に従ってオンライン展開を実行する。このオンライン展開の結果は、上述したコンパイラ１０による展開結果と同じであり、図９Ａに示した手続コードを生成する。また、「SUM(A)」が「SUM_SAMPLE_1(A) 」に置き換えられる動作も上述した通りである。

ただし、コンパイラ３０においては、同じ手続コードを重複して生成することを回避するために、順次、生成した手続コードを呼び出すための呼出の名前と、その手続の特徴が管理テーブルに登録される。なお、「特徴」は、引数の型、引数の次元数、引数の各次元の下限値および上限値である。したがって、

「SUM(A)」についてオンライン展開が実行されると、図１３Ａに示すように、「SUM(A)」に対応するレコードが登録される。

続いて、コンパイラ３０は、「SUM(A2)」の特徴を抽出し、図１３Ｂに示すように、管理テーブルに既に登録されている呼出の特徴とその抽出した特徴とを比較する。ここでは、「SUM_SAMPLE_1(A)」および「SUM(A2)」の特徴が互いに完全に一致しているので、「SUM(A2)」に対しては新たに手続コードを生成することなく、「SUM(A)」に対して生成された手続コードを「SUM(A2)」から呼び出すようにするために、「SUM(A2)」を「SUM_SAMPLE_1(A2)」に変換する。この結果、ソースプログラムの４行目は、以下にように変換されることになる。

B=SUM(A)+SUM(A2)　→　B=SUM_SAMPLE_1(A)+SUM_SAMPLE_1(A2)

したがって、この行が実行されると、「SUM_SAMPLE_1(A)」により図９Ａに示した手続コードが呼び出され、さらに、「SUM_SAMPLE_1(A2)」によってもその同じ手続コードが呼び出されることになる。

なお、「SUM_SAMPLE_1(A)」および「SUM(A2)」の特徴が互いに完全に一致していたので、管理テーブルには新たなレコードは追加されず、管理テーブルは図１３Ａに示す状態を保持する。

続いて、コンパイラ３０は、「SUM(N(51:100))」の特徴を抽出し、図１３Ｃに示すように、管理テーブルに既に登録されている呼出の特徴とその抽出した特徴とを比較する。ここでは、「SUM_SAMPLE_1(A)」及び「SUM(N(51:100))」の特徴が互いに一致していないので、「SUM(N(51:100))」に対応する手続コードを新たに生成する。この手続コードは、図９Ｂに示した手続コードと同じである。

そして、コンパイラ３０は、管理テーブルをサーチし、先に登録されていない名前（ここでは、「SUM_SAMPLE_2」）を決定し、それを「SUM(N(51:100))」に割り当てる。この後、ソースプログラムの５行目が下記のように書き換えられる。この処理は、図５～図９を参照しながら説明した処理と同じである。

WRITE(*,*) SUM(N(51:100)) → WRITE(*,*) SUM_SAMPLE_2(N(51:100))

なお、新たに使用されることとなった名前および対応する特徴は、管理テーブルに登録される。

さらに、コンパイラ３０は、「SUM(M(51:200))」に対して同様の処理を実行する。この場合、図１３Ｄに示すように、「SUM(M(51:200))」の特徴は、先に登録されている「SUM_SAMPLE_2」の特徴と類似しているが、完全には一致していないので、「SUM(M(51:200))」に対応する手続コードを新たに生成する。この新たに生成される手続コードを図１４に示す。

そして、コンパイラ３０は、管理テーブルをサーチし、先に登録されていない１つの名前（「SUM_SAMPLE_3」）を決定し、それを「SUM(M(51:200))」に対して割り当てる。この後、ソースプログラムの６行目が下記のように書き換えられる。

WRITE(*,*) SUM(M(51:200)) → WRITE(*,*) SUM_SAMPLE_3(M(51:200))

このように、コンパイラ３０は、図１２Ａに示すソースプログラムが入力されると、図１２Ｂに示すオブジェクトコード、および図９Ａ、図９Ｂ、および図１４に示す手続コードを出力する。ここで、図９Ａに示す手続コードは、２つの呼出により共有的に呼び出される。すなわち、上記実施例では、ソースプログラムから変換すべき対象が４つ検出されたが、生成された手続コードは３つである。このように、ソースプログラムの中に複数の互いに同じ変換対象が

あった場合、コンパイラ３０の総出力コード量は図５に示したコンパイラ１０と比較して減少する。

なお、図１０～図１４に示した実施例では、図８に示したテンプレートを用いて関数 SUM の手続コードを生成したが、テンプレートの表現の抽象度を高めると、生成される手続コードの数をさらに減少させることができる。

図１５は、図８に示したテンプレートと比較して抽象度の高い表現で作成したテンプレートの例である。図８に示したテンプレートは、引数の型、引数の次元数、および引数の各次元の下限値および上限値をパラメータとして手続を展開するフォーマットであったが、図１５に示すテンプレートは、引数の型および引数の次元数のみをパラメータとして手続を展開するフォーマットである。したがって、図１５に示すテンプレートを用いた場合、引数の型および引数の次元数が互いに同じであれば、引数の各次元の下限値および上限値の不一致は無視され、配列の大きさに因らない手続コードが作成される。

図１６は、図１５に示すテンプレートを用いて変換すべき対象を展開する際に比較されるパラメータを示す図である。ここでは、図１２Ａに示したソースプログラムから検出された４つの変換すべき対象について示している。

図１５に示すテンプレートを用いる場合、比較すべきパラメータは、上述したように、変換すべき対象の「引数の型」および「引数の次元数」である。従って、「SUM(A)」および「SUM(A2) 」に対して互いに同じ手続コードが生成されることになり、同様に、「SUM(N(51:100))」および「SUM(M(51:200))」に対して互いに同じ手続コードが生成されることになる。すなわち、コンパイラ３０は、「SUM(A)」および「SUM(A2) 」に対して１つの手続コードを生成し、同様に、「SUM(N(51:100))」および「SUM(M(51:200))」に対しても１つの手続コ

ードを生成する。このとき、「SUM(A)」および「SUM(A2)」は、互いに同じ手続呼出（ここでは、「SUM_SAMPLE_1」とする。）に変換される。同様に、「SUM(N(51:100))」および「SUM(M(51:200))」も互いに同じ手続呼出（ここでは、「SUM_SAMPLE_2」とする。）に変換される。

図１７は、図１５に示すテンプレートを用いた場合のコンパイラ３０の出力を示す図である。コンパイラ３０は、図１２Ａに示したソースプログラムが入力されると、図１５に示したテンプレートを用いた場合、図１７に示すオブジェクトコード、および２つの手続コード（オンラインコード）を出力する。ここで、手続コードＡは、「SUM_SAMPLE_1」により呼び出され、また、手続コードＢは、「SUM_SAMPLE_2」により呼び出される。

なお、図１７に示す例では省略されているが、実際には、オブジェクトコードにおいて手続コードに対するインタフェース宣言が記述される必要がある。例えば、Fortran90 プログラムとしてオブジェクトコードを生成する場合は、ソースプログラム中の変換すべき対象を新たな手続呼出に変換する際、コンパイラが、その新たな手続呼出に関するインタフェース宣言を生成する。このことは、他の実施例においても同様である。

上記図５および図１０に示した実施例では、コンパイラへの入力が１つのプログラムであることを前提としていたが、実際には、複数のソースプログラムを含むファイルを一括してコンパイルしたい場合が多々ある。以下では、複数のソースプログラムを含むファイルを一括してコンパイルするコンパイラについて説明する。

図１８は、本発明のさらに他の形態のコンパイラの機能ブロック図である。図１８において使用する符号のうち、図５または図１０において先に使用して

いるものは同じ機能のユニットを表す。

コンパイラ４０は、基本的には、図１０に示したコンパイラ３０と同じである。ただし、コンパイラ３０に設けられる重複判定部３１は、各ソースプログラム中に重複する変換すべき対象が存在するか否かを調べるのに対し、コンパイラ４０に設けられる重複判定部４１は、入力ファイル毎に重複する変換すべき対象が存在するか否かを調べる。重複判定部４１による判定結果は、変換部１２および展開部１３に通知される。この通知を受けた変換部１２および展開部１３の動作は、基本的に、図１０を参照しながら説明した通りである。

図１９は、図１８に示したコンパイラの動作を説明するためのフローチャートである。ここでは、コンパイラ４０に入力されるソースファイルをファイルF、コンパイラ４０によりコンパイルされたオブジェクトファイルをファイルF'と呼ぶことにする。なお、以下では、ファイルFは、１以上のソースプログラムP1 ,.....,Pt を含むものとする。また、この処理は、１つのソースファイルが入力されるごとに実行される。

ステップＳ２１～Ｓ３０の各処理は、基本的に、図１１に示したステップＳ１１～Ｓ２０とそれぞれ同じである。ただし、ステップＳ２１～Ｓ３０では、ファイル単位で処理が行われる。すなわち、ステップＳ２２では、ファイルFに含まれている１以上のソースプログラムから変換すべき対象が検出される。また、ステップＳ２５では、変換すべき対象に対してファイルF内でユニークな名前を生成する。

コンパイラ４０は、ソースプログラムP1 ,.....,Pt を含むファイルFが入力されると、上記ステップＳ２１～Ｓ３０の処理を実行することにより、各変換すべき対象がそれぞれ手続の呼出に置き換えられたオブジェクトプログラム

P'1,....,P't を生成し、また、各手続の呼出にそれぞれ対応して手続コードS1 ,....,Sm を生成する。そして、これらのオブジェクトプログラムおよび生成された手続コードは、同一のファイルに出力される。

なお、手続コードは、生成される毎に、順次、１つずつ出力してもよいし、プログラム単位の処理が終了する毎に出力してもよいし、あるいはファイル単位の処理が終了した時点で一括して出力してもよい。また、ソースコードが複数のファイルから構成される場合には、各ファイル毎に上記処理を繰り返し実行すればよい。

このように、コンパイラ４０によれば、複数のソースプログラムを含むファイル内に存在する互いに重複した変換すべき対象に対して、重複して手続コードを生成することなく、必要最小限の手続コードを生成するので、コンパイラの総出力コード量がさらに減少する。

次に、具体的な実施例を説明する。以下では、コンパイラ４０に対して図２０に示すソースファイル（tiny.f）が入力された場合を想定し、関数 SUM の呼出を変換対象として説明する。なお、コンパイラ４０は、図８に示したテンプレートを利用するものとする。

ソースファイルが入力されると、コンパイラ４０は、そのソースファイルに含まれている２つのプログラム（メインプログラムおよびサブプログラム）をスキャンすることにより関数 SUM の呼出を検出する。図２０に示す例においては、関数 SUM は、メインプログラムの「SUM(A)」及び「SUM(N(51:100))」、およびサブプログラムの「SUM(Q)」により呼ばれていることが検出される。

続いて、コンパイラ４０は、検出した各変換すべき対象の特徴（引数の型、引数の次元数など）に基づいて、生成すべき手続コードが同一になるものが存

在するか否かを調べる。ここでは、「SUM(A)」および「SUM(Q)」の特徴が互いに同じである。従って、「SUM(A)」および「SUM(Q)」に対しては、これらにより共有される１つの手続コードが生成される。なお、「SUM(N(51:100))」に対しては、１つの独立した手続コードが生成される。

図２１は、図２０に示すソースファイルが入力されたときのコンパイラ４０の出力である。ここでは、呼出の名前は、手続の名前と、ファイル名と、そのファイルの中での登場順番との組合せにより決定されるものとしている。従って、「SUM(A)」および「SUM(Q)」は「SUM_TINY_1」に置き換えられている。また、「SUM(N(51:100))」は「SUM_TINY_2」に置き換えられている。そして、手続コードＡ及び手続コードＢは、それぞれ「SUM_TINY_1」及び「SUM_TINY_2」により呼び出されることになる。

このように、コンパイラ４０によれば、互いに同じ変換すべき対象が同一ファイル内に存在する場合、それらにより共有される手続コードが生成される。このため、生成した手続コードをオブジェクトプログラムと同じファイルに出力するようにすれば、ファイルの取り扱いが簡単になる。例えば、入力ファイルのファイル名が「tiny.f」であるときは、出力ファイルのファイル名を「tiny.o」とする。そして、そのファイルにオブジェクトプログラムおよび手続コードの双方を出力する。

図２２は、本発明のさらに他の形態のコンパイラの機能ブロック図である。コンパイラ５０は、複数のファイルが一括して入力されることを考慮し、入力された複数のファイルに存在する互いに同じ変換すべき対象を同一の手続呼出に変換すると共に、その手続呼出に対して１つの手続コードを生成する。この手続コードは、複数のファイルにより共通化され、複数の手続呼出により共有

的に呼び出される。

上記機能を実現するために、重複判定部５１は、複数の入力ファイル中に重複する変換すべき対象が存在するか否かを調べ、その判定結果を変換部１２および展開部１３に通知する。この通知を受けた変換部１２および展開部１３の動作は、基本的に、図１０を参照しながら説明した通りである。

リンク編集部２３は、既存の一般的なプログラムにより実現可能であり、コンパイル済みのコードとデータファイルとをリンクさせることにより実行可能プログラムを生成するプログラムである。１つのファイルは、一般に１つ以上のプログラム単位（サブルーチンや関数等）を含み、１つの実行可能ファイルは、１つ以上のファイルのソースコードをコンパイルし、それらをリンク編集して作成される。また、リンク編集部２３は、ライブラリの生成機能も持っている。

図２３は、図２２に示したコンパイラの動作を説明するためのフローチャートである。ここでは、コンパイラ５０に入力される複数のソースファイルをファイルF1 ,....,Fs 、コンパイラ５０によりコンパイルされたオブジェクトファイルをファイルF'1,....,F's と呼ぶことにする。また、各ソースファイルは、それぞれ１以上のソースプログラムを含むものとする。なお、図２３に示す処理は、一連の複数のソースファイルが入力されるごとに実行される。

ステップＳ３１～Ｓ４０の各処理は、基本的に、図１１に示したステップＳ１１～Ｓ２０とそれぞれ同じである。ただし、ステップＳ３１～Ｓ４０では、一連の複数のファイルを単位として処理が行われる。すなわち、ステップＳ３２では、ファイルF1 ,....,Fs に含まれている複数のソースプログラムから変換すべき対象が検出される。また、ステップＳ３５では、変換すべき対象に

対して一連のファイルF1 ,....,Fs 内でユニークな名前を生成する。なお、出力ファイルをユーザが扱う通常のファイルとする場合には、オブジェクトコードと手続コード（オンラインコード）とが対応付けられる。

コンパイラ５０は、ファイルF1 ,....,Fs が入力されると、上記ステップＳ３１～Ｓ４０の処理を実行することにより、各変換すべき対象がそれぞれ手続の呼出に置き換えられたオブジェクトプログラムを生成し、これらのオブジェクトプログラムを含むファイルF'1,....,F's を出力する。また、コンパイラ５０は、各手続の呼出にそれぞれ対応する手続コードを生成し、これらの手続コードを共通ファイルF0 に出力する。

なお、手続コードは、生成される毎に、順次、１つずつ出力してもよいし、プログラム単位の処理が終了する毎あるいはファイル単位の処理が終了する毎に出力してもよい。また、手続コードは、オブジェクトファイルを含むファイルと同じファイルに出力してもよい。また、コンパイラ５０は、出力ファイルを一時ファイルとして管理し、リンク編集処理をも行うことによって実行ファイルを生成してもよい。

このように、コンパイラ５０によれば、複数のソースファイル内に存在する互いに重複した変換すべき対象に対して、重複して手続コードを生成することなく、必要最小限の手続コードを生成するので、コンパイラの総出力コード量がさらに減少する。

次に、具体的な実施例を説明する。以下では、コンパイラ５０に対して図２４に示す２つのソースファイル（tiny1.f およびtiny2.f ）が入力された場合を想定し、関数SUM の呼出を変換対象として説明する。なお、コンパイラ５０は、図８に示したテンプレートを利用するものとする。

ソースファイルが入力されると、コンパイラ５０は、ファイル tiny1.f に含まれているプログラム（メインプログラム）、およびファイル tiny2.f に含まれているプログラム（サブプログラム）をスキャンすることにより関数 SUM の呼出を検出する。図２４に示す例においては、関数 SUM は、ファイル tiny1.f の「SUM(A)」及び「SUM(N(51:100))」、およびファイル tiny2.f 「SUM(Q)」により呼ばれていることが検出される。

続いて、コンパイラ４０は、検出した各変換すべき対象の特徴（引数の型、引数の次元数など）に基づいて、生成すべき手続コードが同一になるものが存在するか否かを調べる。ここでは、「SUM(A)」および「SUM(Q)」の特徴が互いに同じである。従って、「SUM(A)」および「SUM(Q)」に対しては、これらにより共有される１つの手続コードが生成される。なお、「SUM(N(51:100))」に対しては、１つの独立した手続コードが生成される。

図２５は、図２４に示すソースファイルが入力されたときのコンパイラ５０の出力である。ここでは、呼出の名前は、手続の名前と、一連の複数の入力ファイルの中での登場順番との組合せにより決定されるものとしている。したがって、「SUM(A)」及び「SUM(Q)」は「SUM_1 」に置き換えられている。また、「SUM(N(51:100))」は「SUM_2 」に置き換えられている。そして、手続コードＡおよび手続コードＢは、それぞれ「SUM_1 」及び「SUM_2 」により呼び出されることになる。

このように、コンパイラ５０は、手続コードは同時にコンパイルされる一連のファイルにより共有される。なお、リンク編集処理では、オブジェクトファイルにおいて手続コードが格納されているファイルが指定される。このリンク処理は、コンパイラ５０により実行されてもよいし、あるいは、図２２に示し

た構成のように、コンパイラとは別のモジュールにより実行されてもよい。コンパイラによりリンク編集処理が実行される場合には、手続コードを一時ファイルに格納し、コンパイル後にその一時ファイルを削除するようにすれば、ソースとの対応付けにおいて問題が生じにくくなる。

上記実施例では、変換すべき対象として「関数 SUM 」を採り上げたが、他の関数やサブルーチンなどであっても同様に処理される。以下では、図５に示したコンパイラ１０に図２６Ａに示すソースプログラムが入力された場合を想定し、「Ａ＊＊Ｎ」を変換すべき対象とする。「Ａ＊＊Ｎ」は、「ＡのＮ乗」を表す。ここで、「Ａ」は、実数型、複素数型、または整数型のスカラであり、「Ｎ」は、整数型のスカラであるものとする。

コンパイラ１０は、ソースプログラムから「Ａ＊＊Ｎ」を検出すると、それを「ＰＯＷ（Ａ，Ｎ）」に変換することによりオブジェクトプログラムを生成すると共に、「ＰＯＷ（Ａ，Ｎ）」により呼び出されるべき手続コード（オンラインコード）を出力する。この手続の定義は、例えば、「Ｎが３以下のときに掛け算を実行し、Ｎが３よりも大きいときには巾乗演算を実行する。」である。

コンパイラ１０は、「ＰＯＷ（Ａ，Ｎ）」に対応する手続コードを生成する際、以下の３つの特徴を参照する。

特徴１：Ａの型

特徴２：Ｎの値が得られるか否か

特徴３：Ｎの値が得られたとき、その値

コンパイラ１０は、「ＰＯＷ（Ａ，Ｎ）」に対応する手続コードを生成する際に利用するサンプルデータとして図２７Ａ～図２７Ｄ、および図２８Ａおよ

び図２８Ｂに示すテンプレートを備える。これらのテンプレートでは、上記特徴１が抽象化されている。

図２７Ａに示すテンプレートは、「Ｎ」が「０」の場合に対応して記述されている。すなわち、「Ｎ＝０」の場合は、「Ａの０乗」は常に「１」なので、テンプレートには予め「Ｒ＝１」が記述されている。

図２７Ｂ～図２７Ｄに示す各テンプレートは、それぞれ「Ｎ」が「１」から「３」の場合に対応して記述されている。ここでは、「Ｎが小さいときには、巾乗演算を実行するよりも掛け算を実行した方が有利である」という前提に立ち、各テンプレートには、それぞれ掛け算式（「Ｒ＝Ａ」「Ｒ＝Ａ＊Ａ」および「Ｒ＝Ａ＊Ａ＊Ａ」）が予め記述されている。

図２８Ａに示すテンプレートは、「Ｎ」が３よりも大きい場合に対応して記述されている。即ち、テンプレートには、巾乗演算式（「Ｒ＝Ａ＊＊Ａ」）が予め記述されている。

図２８Ｂに示すテンプレートは、ソースプログラムがコンパイルされる際に「Ｎ」の値が不明である場合に対応して記述されている。この場合、テンプレートには、「Ｎ」の値が決まった時点でその値に対応する演算式が選択されるような記述となっている。

コンパイラ１０は、ソースプログラムが入力されると、各行をサーチすることにより「Ａ＊＊Ｎ」という表現を検出する。図２６Ａに示すプログラムが入力された場合、４行目の「２＊＊LEN 」および５行目の「（Ｒ＊２）＊＊２」が検出される。

まず、「２＊＊LEN 」がオンライン展開される。「２＊＊LEN 」は、プログラム SUBROUTINE の中で最初に登場した変換すべき対象なので、「POW_SUBP_

1」という名前が割り当てられる。続いて、上述した変換規則に従って以下の変換を行う。

REAL::S(2**LEN-1) → REAL::S(POW_SUBP_1(2,LEN)-1)

そして、引数を解析することによって以下の情報を得る。

特徴１：型＝INTEGER

特徴２：Ｎの値は得られず

特徴３：なし

コンパイラ１０は、上記得られた情報により適切なテンプレートを選択してオンラインコードを生成する。すなわち、ここでは、「Ｎ」の値が得られていないので、図２８Ｂに示したテンプレートが選択され、図２６Ｂに示すオンラインコードＡが生成される。

同様に、「（Ｒ＊２）＊＊２」がオンライン展開される。「（Ｒ＊２）＊＊２」は、プログラム SUBROUTINE の中で第２番目に登場した変換すべき対象なので、「POW_SUBP_2」という名前が割り当てられ、以下の変換が行われる。

M=PAI*(R*2)**2 → M=PAI*POW_SUBP_2((R*2),2)

そして、引数を解析することによって以下の情報を得る。

特徴１：型＝REAL

特徴２：Ｎの値は得られた

特徴３：Ｎ＝２

コンパイラ１０は、上記得られた情報に従って図２７Ｃに示すテンプレートを選択し、図２６Ｂに示すオンラインコードＢを生成する。そして、コンパイラ１０は、オブジェクトプログラムと共に、オンラインコードＡおよびＢを出力する。

なお、プログラムの実行時にオンラインコードを呼び出すためのオーバーヘッドを減少させるためには、本実施例のコンパレータで生成したオンラインコードを後続のコンパイラ（オブジェクト言語が機械後又はアセンブラ等であるコンパイラ）によりインライン展開してもよい。インライン展開は、上述したように、プリプロセッサ方式のコンパイラで高級言語を対象とすると問題が生じることが多いが、オブジェクト言語が機械後又はアセンブラ等である通常のコンパイラでは、問題なくインライン展開を実行できることが多い。

上記コンパイラの機能は、コンピュータを用いて図６、図１１、図１９、または図２３のフローチャートに示した処理を記述したプログラムを実行することにより実現される。そのプログラムを実行するコンピュータ６０のブロック図を図２９に示す。

ＣＰＵ６１は、図６、図１１、図１９、または図２３のフローチャートに示した処理を記述したプログラムを記憶装置６２からメモリ６３にロードして実行する。記憶装置６２は、例えばハードディスクであり、上記プログラム、および各種テンプレートを格納する。一方、メモリ６３は、例えば半導体メモリであり、ＣＰＵ６１の作業領域として使用される。図１３Ａなどに示した管理テーブルは、このメモリ６３に作成される。

記憶媒体ドライバ６４は、ＣＰＵ６１の指示に従って可搬性記憶媒体６５にアクセスする。可搬性記憶媒体６５は、半導体デバイス（ＩＣカード等）、磁気的作用により情報が入出力される媒体（フロッピーディスク、磁気テープなど）、光学的作用により情報が入出力される媒体（光ディスクなど）を含む。通信制御装置６６は、ＣＰＵ６１の指示に従って網との間でデータを送受信する。

図３０は、本発明に係わるソフトウェアプログラムなど（テンプレートを含む）の提供方法を説明する図である。本発明に係わるプログラムは、例えば、以下の３つの方法の中の任意の方法により提供される。

(a) コンピュータ６０にインストールされて提供される。この場合、プログラム等は、たとえば、出荷前にプレインストールされる。

(b) 可搬性記憶媒体に格納されて提供される。この場合、可搬性記憶媒体６５に格納されているプログラム等は、基本的に、記憶媒体ドライバ６４を介して記憶装置６２にインストールされる。

(c) 網上のサーバから提供される。この場合、基本的には、コンピュータ６０がサーバに格納されているプログラム等をダウンロードすることによってそのプログラム等を取得する。

このように、本実施例のコンパイラは、プリプロセッサ方式であっても、ソースプログラム中における変換すべき対象の出現位置に係わらず、その変換すべき対象を容易に展開することができる。

また、手続呼出をインライン展開できる場合であっても、総出力コード量、およびコンパイル時間の点で本実施例のコンパイラの方が有利である。

(a) 出力コードの総量

図１０、図１８、および図２２に示したコンパイラによれば、互いに同種の変換すべき対象が複数存在する場合には、それらに対して１つのオンラインコードが生成され、複数の手続呼出によりその生成されたオンラインコードが共有的に呼び出されるので、オブジェクトの総量が小さくなる。なお、図５に示したコンパイラの出力の総量は、インライン展開の場合のそれと比較して同程度である。

(b) コンパイル時間

一般に、コンパイラ内での最適化処理に要する時間は、コンパイル単位（１つの関数やサブルーチン等）の大きさの２乗～３乗に比例することが多い。このため、コンパイル単位のコード量が数倍になると、最適化のための処理時間は数十倍になることがある。

図５に示したコンパイラによれば、上述したように、出力コードの総量はインライン展開の場合と同程度だが、展開コード（オンラインコード）は独立したコンパイル単位として分割されて生成されるため、全体のコンパイル時間が短縮される。

また、図１０、図１８、および図２２に示したコンパイラの場合は、生成されるオンラインコードの数が減少するので、コンパイル時間はさらに短縮される。

さらに、本実施例のコンパイラを使用した場合、オンライン展開後、入力コードのコンパイルをオンラインコードのコンパイルよりも先に実行するようにすれば、入力コードのエラーを速く検出することができる。インライン展開を使用した場合には、入力コードが大きくなるので、エラー検出に長い時間を要することが予想される。

産業上の利用可能性

本発明は、プリプロセッサ方式のコンパイラとして広く利用され得る。

請求の範囲

1．入力された第1のプログラムをコンパイルして第2のプログラムを出力するコンパイラ装置であって、

上記第1のプログラムの一部を手続呼出に変換して上記第2のプログラムを作成する変換手段と、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成する生成手段と、

上記第2のプログラムおよび上記生成手段により生成されたコードを出力する出力手段と、

を有するコンパイラ装置。

2．上記第1のプログラムから予め決められた特定のパターンを検出する検出手段をさらに有し、

上記変換手段は、その検出手段により検出されたパターンをそのパターンに対応する手続呼出に変換する請求項1に記載のコンパイラ装置。

3．上記検出手段は、上記第1のプログラムから組込み手続を呼び出すための呼出を検出する請求項2に記載のコンパイラ装置。

4．上記検出手段が上記第1のプログラムから複数の互いに同じパターンを検出した際には、上記変換手段は、それら複数のパターンを同一の手続呼出に変換する請求項1に記載のコンパイラ装置。

5．1以上のプログラムを含む第1のファイルが入力され、その1以上のプログラムをコンパイルし、そのコンパイルされた1以上のプログラムを含む第2のファイルを出力するコンパイラ装置であって、

上記第１のファイルに含まれている１以上のプログラムから予め決められた特定のパターンを検出する検出手段と、

上記検出手段により検出されたパターンをそのパターンに対応する手続呼出に変換することにより上記第１のファイルに含まれている１以上のプログラムをコンパイルする変換手段と、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成する生成手段と、

上記変換手段によりコンパイルされたプログラムおよび上記生成手段により生成されたコードを出力する出力手段とを有し、

上記変換手段が上記第１のファイルから複数の互いに同じパターンを検出した際には、上記変換手段は、それら複数のパターンを同一の手続呼出に変換するコンパイラ装置。

６．上記出力手段は、上記生成手段により生成されたコードを上記第２のファイルに出力する請求項５に記載のコンパイラ装置。

７．各ファイルがそれぞれ１以上のプログラムを含む複数のファイルが入力され、それら入力ファイルに含まれているプログラムをコンパイルするコンパイラ装置であって、

上記入力ファイルに含まれているプログラムから予め決められた特定のパターンを検出する検出手段と、

上記検出手段により検出されたパターンをそのパターンに対応する手続呼出に変換することにより上記入力ファイルに含まれているプログラムをコンパイルする変換手段と、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成す

る生成手段と、

上記変換手段によりコンパイルされたプログラムおよび上記生成手段により生成されたコードを出力する出力手段とを有し、

上記変換手段が上記入力された複数のファイルから複数の互いに同じパターンを検出した際には、上記変換手段は、それら複数のパターンを同一の手続呼出に変換するコンパイラ装置。

8. 入力された第1のプログラムをコンパイルして第2のプログラムを出力するコンパイル方法であって、

上記第1のプログラムの一部を手続呼出に変換して上記第2のプログラムを作成するステップと、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成するステップと、

上記第2のプログラムおよび上記生成手段により生成されたコードを出力するステップと、

を有するコンパイル方法。

9. コンピュータに、入力された第1のプログラムをコンパイルして第2のプログラムを出力させる、ためのプログラムを格納する記憶媒体であって、

上記第1のプログラムの一部を手続呼出に変換して上記第2のプログラムを作成させ、

上記手続呼出により呼び出されるべき手続の定義を記述したコードを生成させ、

上記第2のプログラムおよび上記生成手段により生成されたコードを出力させるプログラムを格納する記憶媒体。

要約書

コンパイラ（１０）は、検出部（１１）、変換部（１２）、展開部（１３）を有する。検出部（１１）は、入力されたソースプログラムから予め決められた対象を検出する。変換部（１２）は、検出部（１１）によって検出された対象を手続呼出に変換する。展開部（１３）は、検出部（１２）により得られた手続呼出によって呼び出されるべき手続の定義を記述したオンラインコードを生成する。コンパイラ（１０）は、検出部（１１）によって検出された対象が呼出手続に置き換えられたプログラム、およびその呼出手続に対応するオンラインコードを出力する。

```
1       SUBROUTINE SUB(A,N)
2       INTEGER N
3       REAL A(ABS(N))
4       WRITE(*,*) A
5       END SUBROUTINE
```

図 1A

```
1       SUBROUTINE SUB(A,N)
2       INTEGER N
        IF (N.GE.0) THEN          ! 展開コード
          TMP = N                 ! 展開コード
        ELSE                      ! 展開コード
          TMP = -N                ! 展開コード
        END IF                    ! 展開コード
3       REAL A(TMP)
4       WRITE(*,*) A
5       END SUBROUTINE
```

図 1B

```
 1      char *copy_string(char *s)
 2      {
 3        int i;
 4        char *buffer = (char*)malloc(strlen(s) + 1);
 5
 6        for (i = 0; s[i] != '\0'; ++i)
 7          buffer[i] = s[i];
 8
 9        return buffer;
10      }
```

図 2A

```
 1      char *copy_string(char *s)
 2      {
 3        int i;
  char *p; /* 展開コード */
  int tmp; /* 展開コード */
  tmp = 0; /* 展開コード */
  for (p = s; *p != '\0'; ++p) /* 展開コード */
    ++tmp; /* 展開コード */
 4        char *buffer = (char*)malloc(tmp + 1);
 5
 6        for (i = 0; s[i] != '\0'; ++i)
 7          buffer[i] = s[i];
 8
 9        return buffer;
10      }
```

図 2B

```
1       IF (Z.GT.EPS) THEN
2         A=B1
3       ELSE IF(ABS(Z).LE.EPS) THEN
4         A=B2
5       ELSE
6         A=B3
7       END IF
```

図 3A

```
1       IF (Z.GT.EPS) THEN
2         A=B1
3a      ELSE
          IF (Z.GE.0.0) THEN        ! 展開コード
            TMP = Z                 ! 展開コード
          ELSE                      ! 展開コード
            TMP = -Z                ! 展開コード
          END IF                    ! 展開コード
3b        IF(TMP.LE.EPS) THEN
4           A=B2
5         ELSE
6           A=B3
3c        END IF
7       END IF
```

図 3B

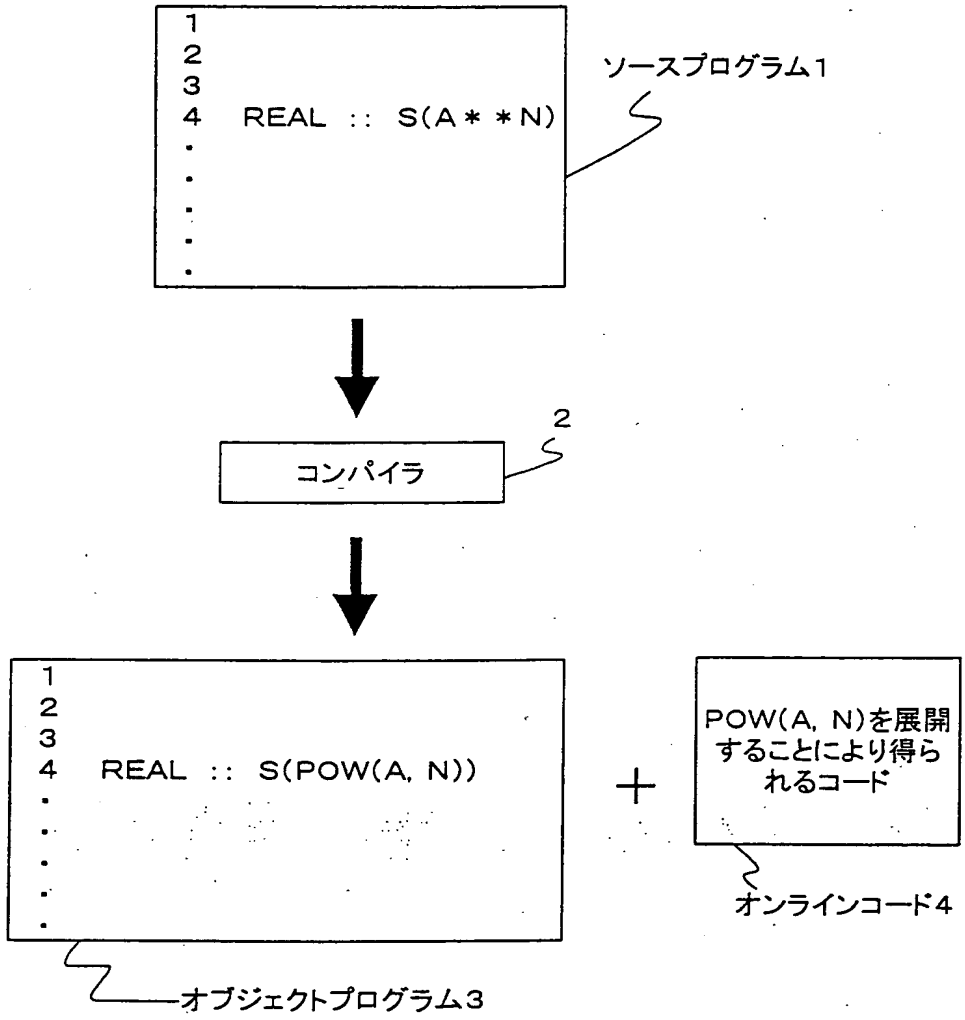

1
2
3
4 REAL :: S(A**N)
ソースプログラム1
2
コンパイラ
1
2
3
4 REAL :: S(POW(A, N))
オブジェクトプログラム3
+
POW(A, N)を展開することにより得られるコード
オンラインコード4

図4

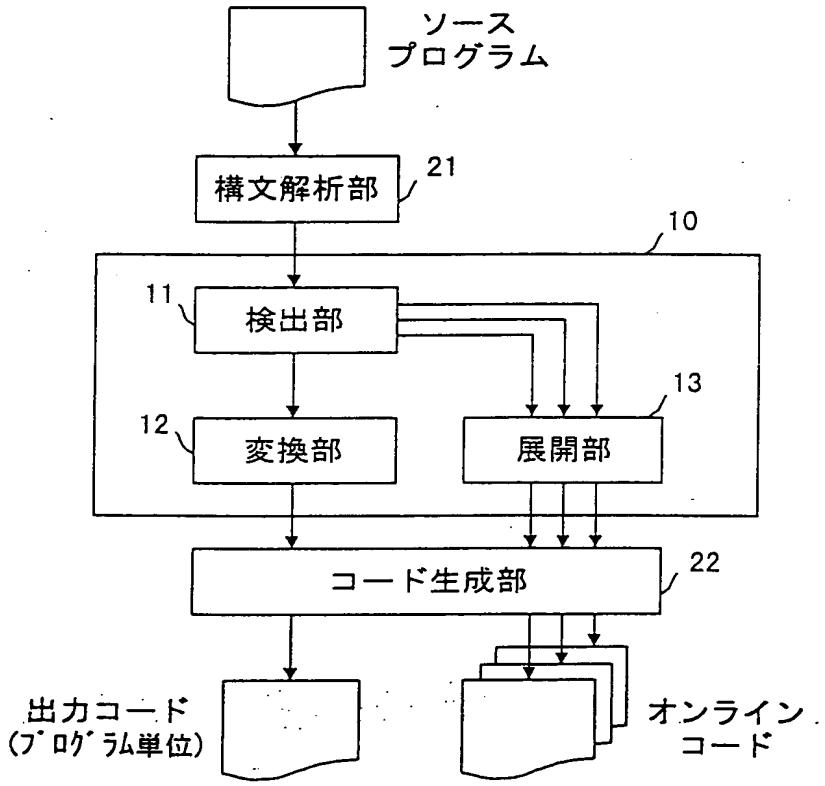

ソース
プログラム
構文解析部
21
10
11
検出部
13
12
変換部
展開部
コード生成部
22
出力コード
(ﾌﾟﾛｸﾞﾗﾑ単位)
オンライン
コード

図5

入力：プログラム単位Ｐ
出力：Ｐを修正したＰ'と、手続Ｓ1,...,Ｓn(0≦ｎ)

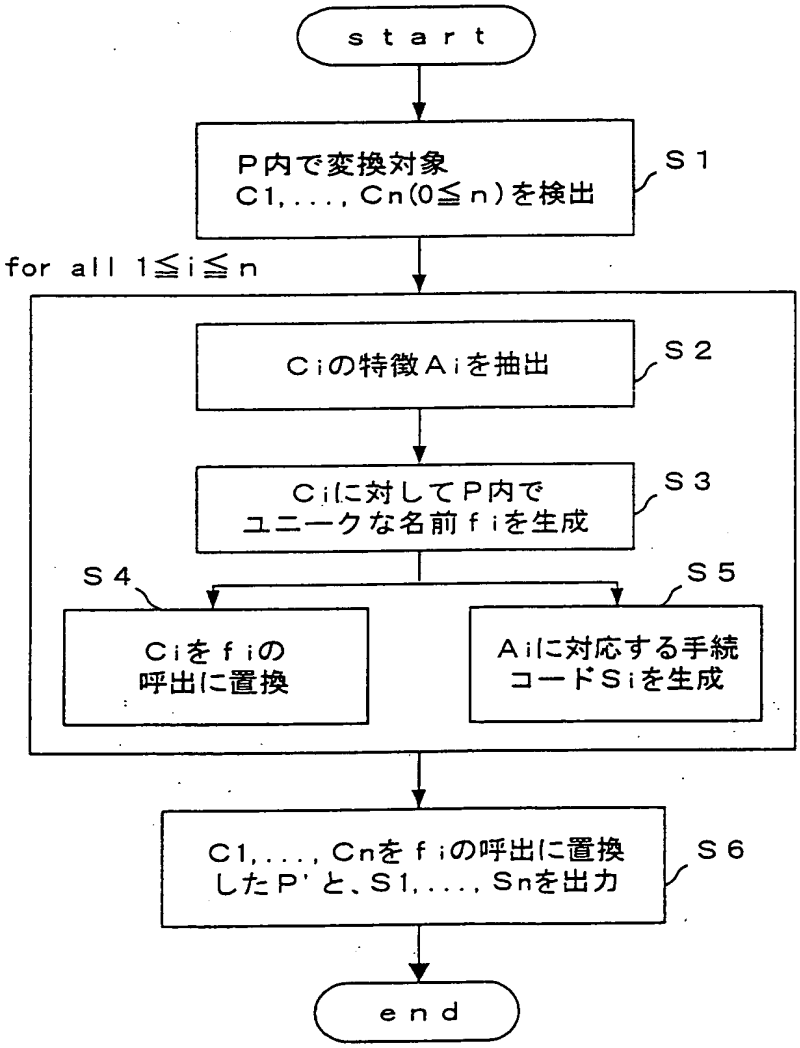


図６

```
1       PROGRAM SAMPL
2       INTEGER N(100)
3       REAL A(10,20,30),B
        ...
4       B = SUM(A)
5       WRITE(*,*) SUM(N(51:100))
6       END
```

図 7A

```
1       PROGRAM SAMPL
2       INTEGER N(100)
3       REAL A(10,20,30),B
        ...
4       B = SUM_SAMPL_1(A)
5       WRITE(*,*) SUM_SAMPL_2(N(51:100))
6       END
```

図 7B

```
      arg-type FUNCTION SUM(X)
      arg-type X(lb(1):ub(1), ···, lb(m):ub(m))
      SUM = 0
      DO 999 Im = lb(m), ub(m)
        :
      DO 999 I1 = lb(1), ub(1)
       SUM = SUM+X(I1,···,Im)
999   CONTINUE
      RETURN
      END
```

図 8

```
      REAL FUNCTION SUM_SAMPL_1(X)
      REAL X(1:10,1:20,1:30)
      SUM_SAMPL_1 = 0
      DO 999 I3 = 1, 30
      DO 999 I2 = 1, 20
      DO 999 I1 = 1, 10
       SUM_SAMPL_1 = SUM_SAMPL_1+X(I1,I2,I3)
  999 CONTINUE
      RETURN
      END
```

図 9A

```
      INTEGER FUNCTION SUM_SAMPL_2(X)
      INTEGER X(51:100)
      SUM_SAMPL_2 = 0
      DO 999 I1 = 51, 100
       SUM_SAMPL_2 = SUM_SAMPL_2+X(I1)
  999 CONTINUE
      RETURN
      END
```

図 9B

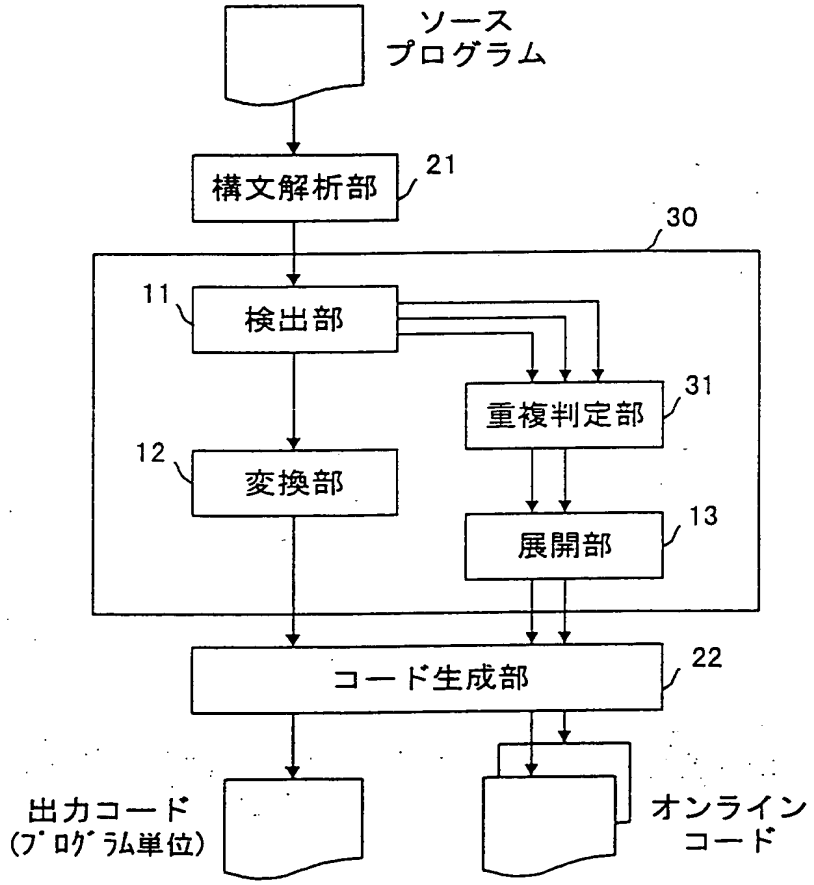

ソース
プログラム
構文解析部
21
30
11
検出部
31
重複判定部
12
変換部
13
展開部
22
コード生成部
出力コード
(ﾌﾟﾛｸﾞﾗﾑ単位)
オンライン
コード

図１０

入力：プログラム単位Ｐ
出力：Ｐを修正したＰ'と、手続Ｓ1,...,Ｓm(0≦m≦n)

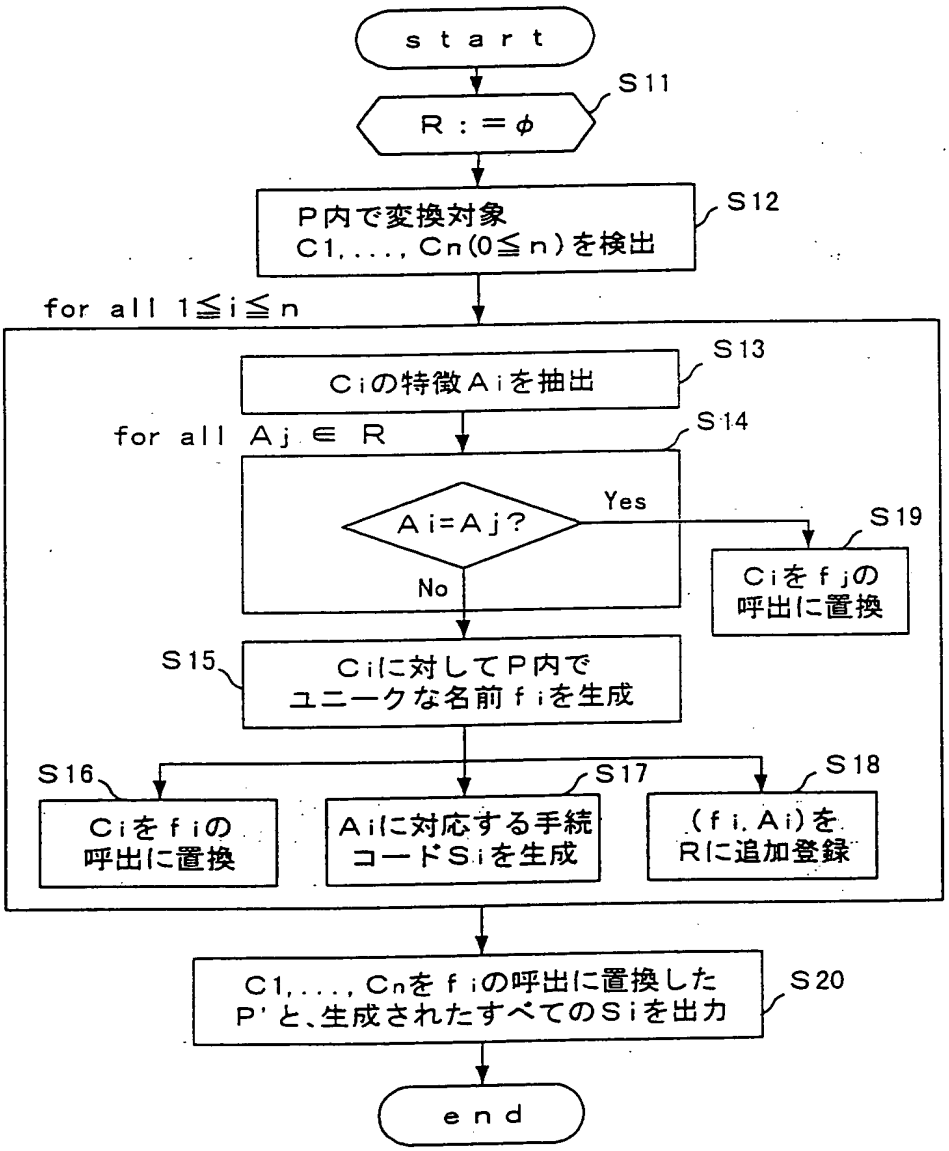


図１１

```
1       PROGRAM SAMPL
2       INTEGER N(100),M(200)
3       REAL A(10,20,30),A2(10,20,30),B
        ...
4       B = SUM(A)+SUM(A2)
5       WRITE(*,*) SUM(N(51:100))
6       WRITE(*,*) SUM(M(51:200))
7       END
```

図 12 A

```
1       PROGRAM SAMPL
2       INTEGER N(100),M(200)
3       REAL A(10,20,30),A2(10,20,30),B
        ...
4       B = SUM_SAMPL_1(A)+SUM_SAMPL_1(A2)
5       WRITE(*,*) SUM_SAMPL_2(N(51:100))
6       WRITE(*,*) SUM_SAMPL_3(M(51:200))
7       END
```

図 12 B

呼出し	*arg−type*	*m*	*lb(1)*	*ub(1)*	*lb(2)*	*ub(2)*	*lb(3)*	*ub(3)*
SUM__SAMPL__1	REAL	3	1	10	1	20	1	30

図13A

呼出し	*arg−type*	*m*	*lb(1)*	*ub(1)*	*lb(2)*	*ub(2)*	*lb(3)*	*ub(3)*
SUM__SAMPL__1	REAL	3	1	10	1	20	1	30
当該呼出	REAL	3	1	10	1	20	1	30

図13B

呼出し	*arg−type*	*m*	*lb(1)*	*ub(1)*	*lb(2)*	*ub(2)*	*lb(3)*	*ub(3)*
SUM__SAMPL__1	REAL	3	1	10	1	20	1	30
当該呼出	INTEGER	1	51	100	—	—	—	—

図13C

呼出し	*arg−type*	*m*	*lb(1)*	*ub(1)*	*lb(2)*	*ub(2)*	*lb(3)*	*ub(3)*
SUM__SAMPL__1	REAL	3	1	10	1	20	1	30
SUM__SAMPL__2	INTEGER	1	51	100	—	—	—	—
当該呼出	INTEGER	1	51	200	—	—	—	—

図13D

```
      INTEGER FUNCTION SUM_SAMPL_3(X)
      INTEGER X(51:200)
      SUM_SAMPL_3 = 0
      DO 999 I1 = 51, 200
       SUM_SAMPL_3 = SUM_SAMPL_3+X(I1)
  999 CONTINUE
      RETURN
      END
```

図 14

```
      arg-type FUNCTION SUM(X)
      arg-type X(:,···,:)
                 m個   ← 抽象化
      SUM = 0
      DO 999 Im = LBOUND(X,m), UBOUND(X,m)
        ⋮
      DO 999 I1 = LBOUND(X,1), UBOUND(X,1)
       SUM = SUM+X(I1,···,Im)
999   CONTINUE
      RETURN
      END
```

図 15

呼出し	*arg−type*	*m*
SUM(A)	REAL	3
SUM(A2)	REAL	3
SUM(N(51:100))	INTEGER	1
SUM(M(51:200))	INTEGER	1

図16

```
      PROGRAM SAMPL
      INTEGER N(100),M(200)
      REAL A(10,20,30),A2(10,20,30),B
      ...
      B = SUM_SAMPL_1(A)+SUM_SAMPL_1(A2)
      WRITE(*,*) SUM_SAMPL_2(N(51:100))
      WRITE(*,*) SUM_SAMPL_2(M(51:200))
      END
```

オブジェクトコード

```
      REAL FUNCTION SUM_SAMPL_1(X)
      REAL X(:,:,:)
      SUM_SAMPL_1 = 0
      DO 999 I3 = LBOUND(X,3),UBOUND(X,3)
      DO 999 I2 = LBOUND(X,2),UBOUND(X,2)
      DO 999 I1 = LBOUND(X,1),UBOUND(X,1)
       SUM_SAMPL_1 = SUM_SAMPL_1+X(I1,I2,I3)
  999 CONTINUE
      RETURN
      END
```

手続コードA

```
      INTEGER FUNCTION SUM_SAMPL_2(X)
      INTEGER X(:)
      SUM_SAMPL_2 = 0
      DO 999 I1 = LBOUND(X,1),UBOUND(X,1)
       SUM_SAMPL_2 = SUM_SAMPL_2+X(I1)
  999 CONTINUE
      RETURN
      END
```

手続コードB

図 17

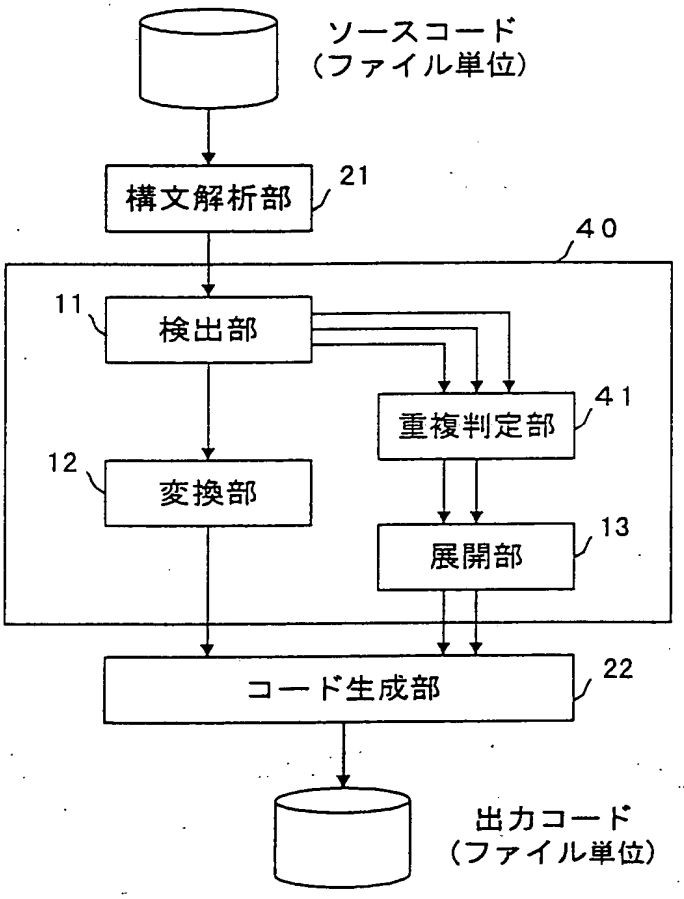

ソースコード
(ファイル単位)
構文解析部
21
40
11
検出部
41
重複判定部
12
変換部
13
展開部
22
コード生成部
出力コード
(ファイル単位)

図１８

クレーム３
入力：プログラム単位Ｐ1,...,Ｐｔ(1≦t)を含むファイルＦ
出力：Ｐ1,...,Ｐｔをそれぞれ修正したＰ1',...,Ｐｔ'と
手続Ｓ1,...,Ｓｍ(0≦m≦n)を含むファイルＦ'

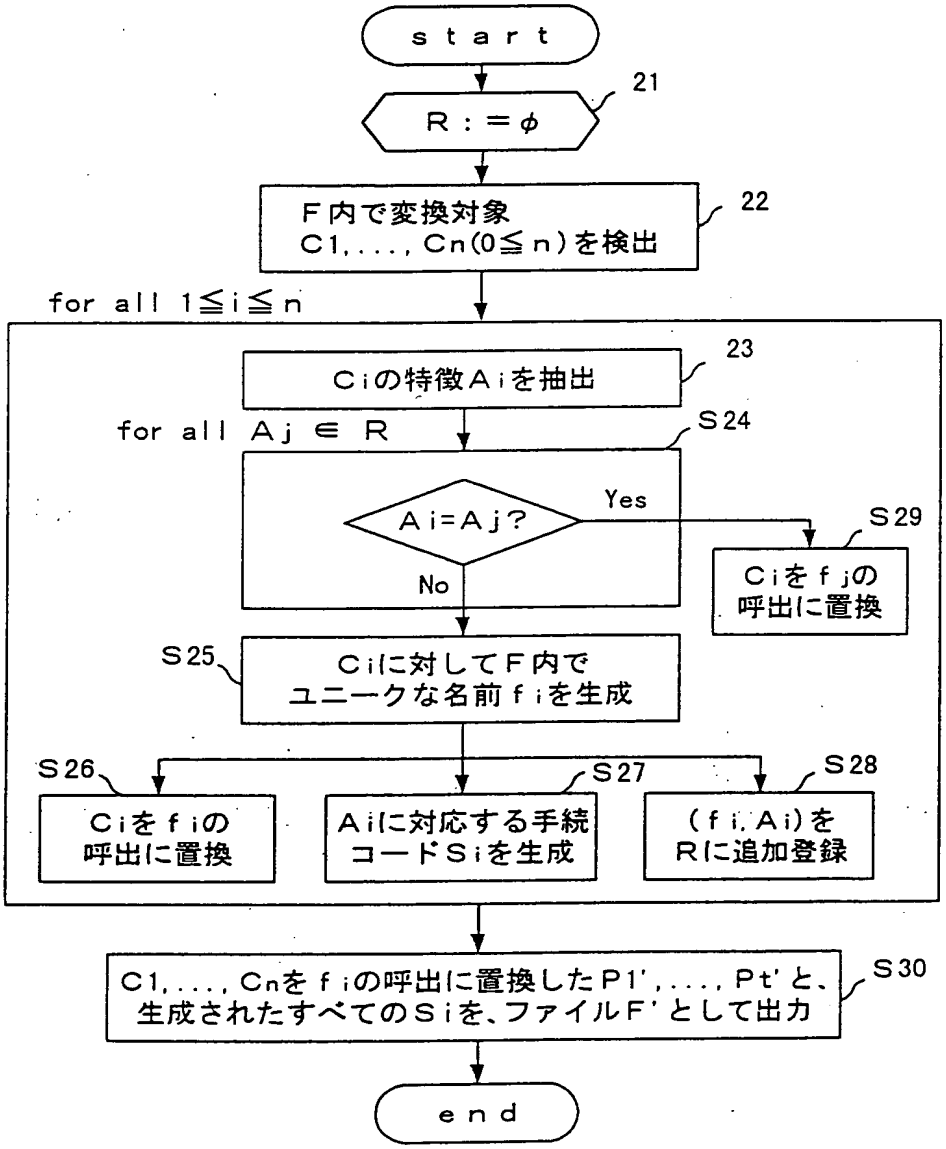


図１９

```
C-- main program ----
      PROGRAM SAMPL
      INTEGER N(100)
      REAL A(10,20,30),A2(10,20,30),B
      ...
      B = SUM(A)
      B = SUM_AND_ADD(A,B)
      WRITE(*,*) SUM(N(51:100))
      END
C-- subprogram ----
      REAL FUNCTION SUM_AND_ADD(Q,S)
      REAL Q(10,20,30),S
      SUM_AND_ADD = SUM(Q)+S
      RETURN
      END
C-- end of user programs ----
```

```
C-- main program ----
      PROGRAM SAMPL
      INTEGER N(100)
      REAL A(10,20,30),A2(10,20,30),B
      ...
      B = SUM_TINY_1(A)
      B = SUM_AND_ADD(A,B)
      WRITE(*,*) SUM_TINY_2(N(51:100))
      END
C-- subprogram ----
      REAL FUNCTION SUM_AND_ADD(Q,S)
      REAL Q(10,20,30),S
      SUM_AND_ADD = SUM_TINY_1(Q)+S
      RETURN
      END
C-- end of user programs ----

      REAL FUNCTION SUM_TINY_1(X)
      REAL X(1:10,1:20,1:30)
      SUM_TINY_1 = 0
      DO 999 I3 = 1, 30
      DO 999 I2 = 1, 20
      DO 999 I1 = 1, 10
       SUM_TINY_1 = SUM_TINY_1+X(I1,I2,I3)
  999 CONTINUE
      RETURN
      END

      INTEGER FUNCTION SUM_TINY_2(X)
      INTEGER X(51:100)
      SUM_TINY_2 = 0
      DO 999 I1 = 51, 100
       SUM_TINY_2 = SUM_TINY_2+X(I1)
  999 CONTINUE
      RETURN
      END
```

手続コードA

手続コードB

図 21

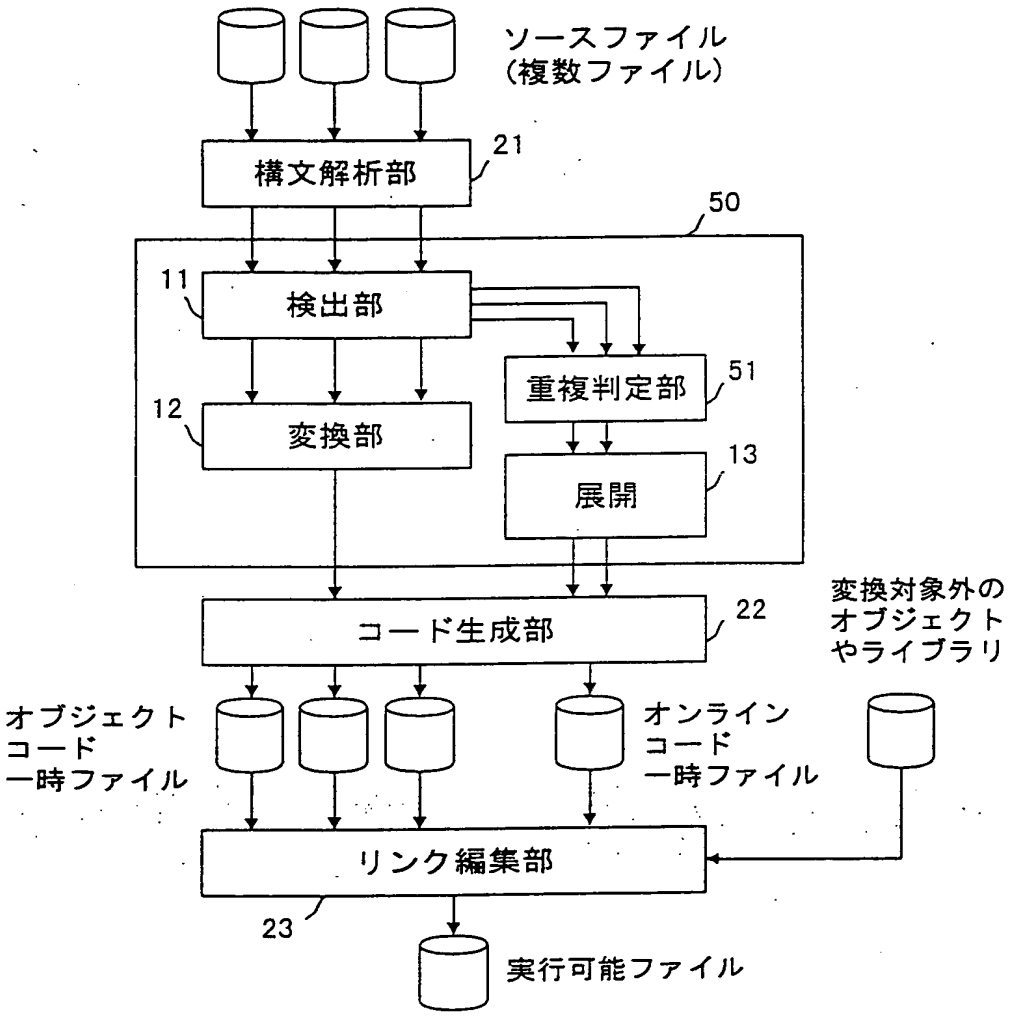

ソースファイル
(複数ファイル)
構文解析部
21
50
11
検出部
重複判定部
51
12
変換部
13
展開
コード生成部
22
変換対象外の
オブジェクト
やライブラリ
オブジェクト
コード
一時ファイル
オンライン
コード
一時ファイル
リンク編集部
23
実行可能ファイル

図２２

クレーム４
入力：プログラム単位Ｐ1,...,Ｐｔ(1≦t)
　　を含むファイルＦ1,...,Ｆｓ(1≦s)
出力：Ｆ1,...,Ｆｓをそれぞれ修正したＦ1',...,Ｆｓ'と
　　手続Ｓ1,...,Ｓｍ(0≦m≦n)を含むファイルＦ0

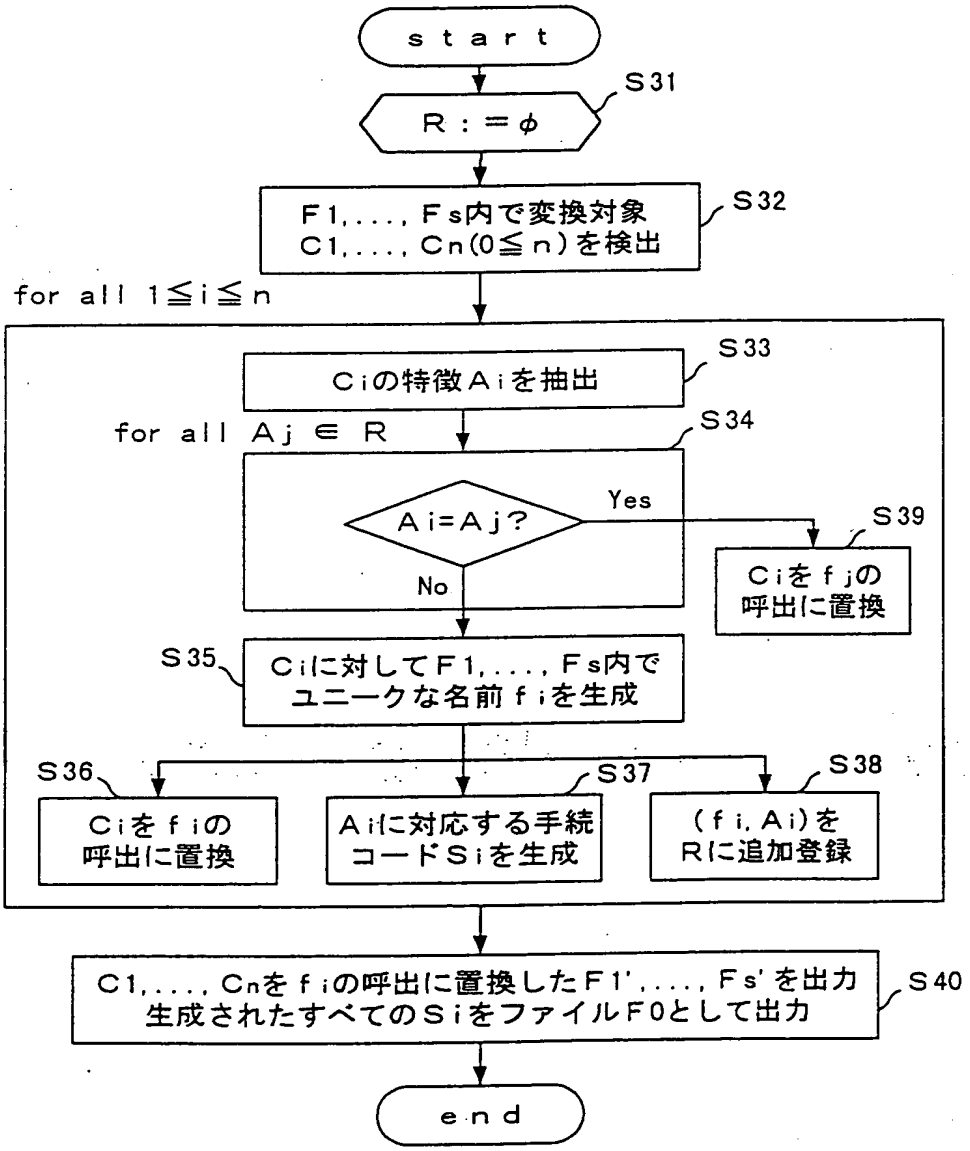


図２３

ファイル tiny1.f:

```
C-- main program ----
      PROGRAM SAMPL
      INTEGER N(100)
      REAL A(10,20,30),A2(10,20,30),B
      ...
      B = SUM(A)
      B = SUM_AND_ADD(A,B)
      WRITE(*,*) SUM(N(51:100))
      END
C-- end of main program ----
```

ファイル tiny2.f:

```
C-- subprogram ----
      REAL FUNCTION SUM_AND_ADD(Q,S)
      REAL Q(10,20,30),S
      SUM_AND_ADD = SUM(Q)+S
      RETURN
      END
C-- end of subprogram ----
```

図 24

ファイル tiny1.o:

```
C-- main program ----
      PROGRAM SAMPL
      INTEGER N(100)
      REAL A(10,20,30),A2(10,20,30),B
      ...
      B = SUM_1(A)
      B = SUM_AND_ADD(A,B)
      WRITE(*,*) SUM_2(N(51:100))
      END
C-- end of main program ----
```

ファイル tiny2.o:

```
C-- subprogram ----
      REAL FUNCTION SUM_AND_ADD(Q,S)
      REAL Q(10,20,30),S
      SUM_AND_ADD = SUM_1(Q)+S
      RETURN
      END
C-- end of subprogram ----
```

ファイル onlines.o:

```
      REAL FUNCTION SUM_1(X)
      REAL X(1:10,1:20,1:30)
      SUM_1 = 0
      DO 999 I3 = 1, 30
      DO 999 I2 = 1, 20
      DO 999 I1 = 1, 10
       SUM_1 = SUM_1+X(I1,I2,I3)
  999 CONTINUE
      RETURN
      END

      INTEGER FUNCTION SUM_2(X)
      INTEGER X(51:100)
      SUM_2 = 0
      DO 999 I1 = 51, 100
       SUM_2 = SUM_2+X(I1)
  999 CONTINUE
      RETURN
      END
```

手続コードA

手続コードB

```
1       SUBROUTINE SUBP(LEN)
2       REAL,PARAMETER :: PAI=3.14159, R=100.0
3       INTEGER LEN,M
4       REAL :: S(2**LEN-1)
        ...
5       M=PAI*(R*2)**2
        ...
6       END SUBROUTINE
```

図 26A

```
SUBROUTINE SUBP(LEN)
REAL,PARAMETER :: PAI=3.14159, R=100.0
INTEGER LEN,M
REAL :: S(POW_SUBP_1(2,LEN)-1)
...
M=PAI*POW_SUBP_2((R*2),2)
...
END SUBROUTINE
```

オブジェクトプログラム

```
FUNCTION POW_SUBP_1(A,N) RESULT(R)
INTEGER A,R
INTEGER N

SELECT CASE (N)
CASE (0)
  R=1
CASE (1)
  R=A
CASE (2)
  R=A*A
CASE (3)
  R=A*A*A
CASE DEFAULT
  R=A**N
END SELECT
RETURN
END FUNCTION
```

オンラインコードA

```
FUNCTION POW_SUBP_2(A,N) RESULT(R)
REAL A,R
INTEGER N

R=A*A
RETURN
END FUNCTION
```

オンラインコードB

図 26B

☒ 27A

```
FUNCTION name(A,N) RESULT(R)
arg-type A,R
INTEGER N

R=1
RETURN
END FUNCTION
```

☒ 27B

```
FUNCTION name(A,N) RESULT(R)
arg-type A,R
INTEGER N

R=A
RETURN
END FUNCTION
```

☒ 27C

```
FUNCTION name(A,N) RESULT(R)
arg-type A,R
INTEGER N

R=A*A
RETURN
END FUNCTION
```

☒ 27D

```
FUNCTION name(A,N) RESULT(R)
arg-type A,R
INTEGER N

R=A*A*A
RETURN
END FUNCTION
```

```
FUNCTION name(A,N) RESULT(R)
arg-type A,R
INTEGER N

R=A**N
RETURN
END FUNCTION
```

図 28A

```
FUNCTION name(A,N) RESULT(R)
arg-type A,R
INTEGER N

SELECT CASE (N)
CASE (0)
  R=1
CASE (1)
  R=A
CASE (2)
  R=A*A
CASE (3)
  R=A*A*A
CASE DEFAULT
  R=A**N
END SELECT
RETURN
END FUNCTION
```

図 28B

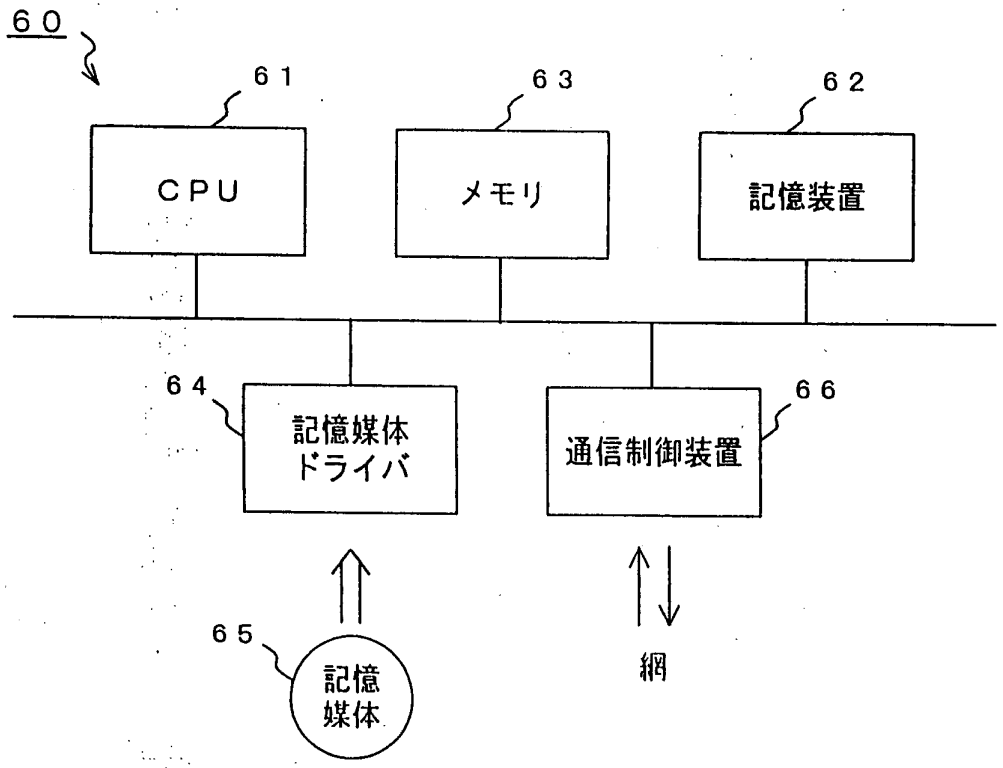


図２９

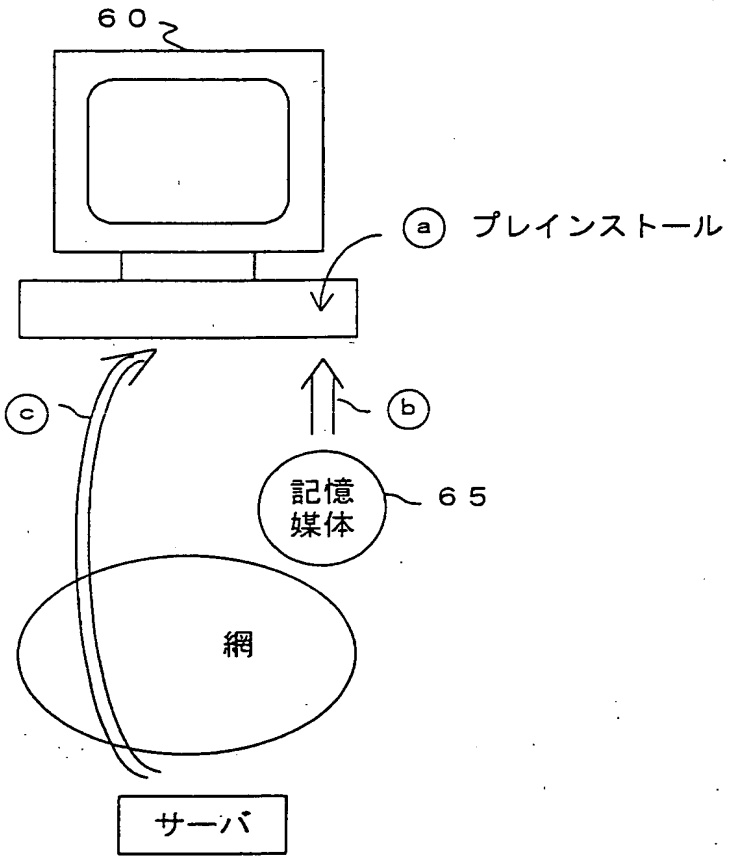


図３０

この謄本は原本と相違ないことを認証する。
平成 13 年 3. 月 12. 日

経済産業事務官　清寺貴幸